НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Навчально-науковий інститут атомної та теплової енергетики

Кафедра інженерії програмного забезпечення в енергетиці

"На правах рукопису"
УДК 004.4

ДО ЗАХИСТУ ДОПУЩЕНО
В.о. завідувача кафедри
_____ Олександр КОВАЛЬ
“___”_____2024р.

Магістерська дисертація

на здобуття ступеня магістра
за освітньо-професійною програмою
«Інженерія програмного забезпечення інтелектуальних кібер-фізичних систем в енергетиці»
зі спеціальності 121 Інженерія програмного забезпечення
на тему: «Програмний застосунок класифікації поведінки водія на дорозі»

Виконала: студентка 2 курсу, групи ТВ-з21мп
Ліла Владислава Олександрівна _____
(прізвище, ім’я, по батькові) (підпис)

Науковий керівник:
професор кафедри ІПЗЕ, доц., д.т.н., Недашківський О.Л. _____
(посада, вчене звання, науковий ступінь, ім’я ПРІЗВИЩЕ) (підпис)

Консультант з постановки завдання:
доцент кафедри ІПЗЕ, доц., к.е.н., Гусєва І. І _____
(посада, вчене звання, науковий ступінь, ім’я ПРІЗВИЩЕ) (підпис)

Рецензент:
професор кафедри ЦТЕ, проф., д.т.н., Отрох С. І _____
(посада, вчене звання, науковий ступінь, ім’я ПРІЗВИЩЕ) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ — 2024

Національний технічний університет України
“Київський політехнічний інститут ім. Ігоря Сікорського”

Навчально-науковий інститут атомної та теплової енергетики
Кафедра інженерії програмного забезпечення в енергетиці
Рівень вищої освіти другий, магістерський
Спеціальність 121 Інженерія програмного забезпечення
Освітньо-професійна програма «Інженерія програмного забезпечення інтелектуальних кібер - фізичних систем в енергетиці»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ Олександр КОВАЛЬ
(підпис)

«____» _____ 2024 р.

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ ДИСЕРТАЦІЮ СТУДЕНТУ

Лілі Владиславі Олександрівні
(прізвище, ім’я, по батькові)

1. Тема дисертації «Програмний застосунок класифікації поведінки водія на дорозі»
Науковий керівник Недашківський Олексій Леонідович д.т.н.. доцент
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від “08” листопада 2023 року №5202-с
2. Строк подання студентом дисертації 10 січня 2024 року
3. Об’єкт дослідження реєстарція, розпізнавання та класифікація поведінки водія на дорозі.
4. Предмет дослідження програмний застосунок класифікації поведінки водія на дорозі.
5. Перелік питань, які потрібно розробити: проаналізувати технології для вирішення проблеми класифікації поведінки водія, проаналізувати технології необхідні для реалізації системи, спроектувати архітектуру системи, розробити програмний застосунок за визначеними методологіями
6. Орієнтований перелік ілюстративного матеріалу архітектура системи, User Flow діаграма, діаграма бази даних, інтерфейс розробленої системи.

7. Консультанти розділів дисертації

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Постановка завдання	доц., к.е.н., Гусєва І. І		

8. Дата видачі завдання «01» листопада 2022 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання магістерської дисертації	строки виконання етапів магістерської дисертації	Примітка
1	Отримання завдання	01.11.2022	виконано
2	Дослідження предметної області	01.11.2022 - 01.12.2022	виконано
3	Постановка вимог до проєктування системи	01.12.2022 - 15.12.2022	виконано
4	Дослідження існуючих рішень	15.12.2022 - 03.01.2023	виконано
6	Розробка програмного продукту	03.03.2023 - 20.09.2023	виконано
7	Тестування	20.09.2023 - 15.10.2023	виконано
8	Захист програмного продукту	16.10-2023 – 20.10.2023	виконано
9	Підготовка магістерської дисертації	21.10.2023 – 17.11.2023	виконано
10	Передзахист	18.12.2023 - 22.12.2023	виконано
11	Захист	15.01.2024 – 19.01.2024	виконано

Студент _____ Владислава ЛІЛА
(підпис) (ім'я ПРІЗВИЩЕ)

Науковий керівник _____ Олексій НЕДАШКІВСЬКИЙ
(підпис) (ім'я ПРІЗВИЩЕ)

РЕФЕРАТ

Структура та обсяг дисертації. Магістерська дисертація складається зі вступу, п`яти розділів, висновку, переліку посилань з 22 найменувань, містить 20 рисунків, 9 таблиць та 4 формули. Повний обсяг магістерської дисертації складає 85 сторінки, з яких перелік посилань займає 2 сторінки, додатки займають 38 сторінки.

Актуальність теми. В сучасних умовах, коли швидкість життя набуває все більшого темпу, питання поведінки водіїв на дорозі стає актуальною та важливою проблемою для забезпечення безпеки та ефективності дорожнього руху. Поведінка водіїв, в значній мірі, визначає рівень ризику дорожньо-транспортних пригод, ефективність транспортних потоків та загальний стан дорожньої інфраструктури.

Однією з найбільш серйозних проблем у цьому контексті є несприятлива поведінка водіїв, яка включає в себе такі аспекти, як порушення правил дорожнього руху, небажані агресивні дії, неправильне використання технічних засобів безпеки, втома та вплив шкідливих речовин.

Ця проблема особливо гостро стоїть перед Україною, де великий обсяг транспортного руху та недостатня розвиненість інфраструктури часто створюють умови для конфліктів та небезпечних ситуацій на дорозі. Водії, порушуючи правила, можуть викликати не лише негативні наслідки для себе, але й стати причиною тяжких дорожньо-транспортних пригод, які часто супроводжуються людськими травмами та втратами.

Дослідження та класифікація поведінки водіїв на дорозі в Україні стає ключовим етапом в розумінні цієї проблеми та розробці ефективних стратегій для її вирішення. Аналіз та розуміння факторів, що впливають на поведінку водіїв, може сприяти розробці превентивних заходів, підвищенню рівня свідомості водіїв та покращенню систем безпеки на дорогах, забезпечуючи тим самим більшу безпеку для учасників дорожнього руху в Україні.

Метою роботи є розробка програмного продукту класифікації поведінки водія на дорозі за допомогою використання сенсорів мобільного телефону.

Об’єктом дослідження є процес класифікації поведінки водія на дорозі та формування оцінки поїздки.

Предметом дослідження є система, що збирає дані із сенсорів мобільного телефону під час поїздки, класифікує поведінку водія на дорозі, на основі зібраних даних, і формує оцінку поїздки по заданим критеріям для зручності користувача.

Методи дослідження. Для розробки системи було використано технології, що дозволяють читати дані із сенсорів мобільного телефону, працювати з картами та здійснювати машинне навчання, що вирішує проблему класифікації: мова програмування Dart, фреймворк Flutter, Google Maps API, бібліотека ml_algo, СУБД SQLite та середовище розробки Android Studio.

Практичне значення полягає в можливості використання системи під час реальних поїздок транспортними засобами, що дозволить зібрати дані із сенсорів телефона про характер руху та визначити поведінку водія на дорозі, надавши йому детальну оцінку поїздки. Таке застосування системи надасть користувачу можливість проаналізувати свої помилки при водінні та покращити навики, що сприятиме зменшенню кількості ДТП на дорогах.

Ключові слова: Bloc, Cubit, багаторівнева архітектура, сенсори, управління станом, машинне навчання, алгоритми класифікації, дерево рішень.

ABSTRACT

The structure and scope of the master's thesis. The master's thesis consists of an introduction, five chapters, a conclusion, a list of references from 22 titles, contains 20 figures, 9 tables and 4 formulas. The full volume of the master's thesis is 126 pages, of which the list of references occupies 85 pages, the appendices occupy 38 pages.

Significance of the topic. In modern conditions, when the speed of life is gaining more and more speed, the issue of the behavior of drivers on the road becomes an urgent and important problem for ensuring the safety and efficiency of road traffic. The behavior of drivers, to a large extent, determines the level of risk of traffic accidents, the efficiency of traffic flows and the general condition of the road infrastructure.

One of the most serious problems in this context is the adverse behavior of drivers, which includes aspects such as violations of traffic rules, unwanted aggressive actions, incorrect use of technical safety devices, fatigue and the influence of intoxicating substances.

This problem is particularly acute in Ukraine, where a large volume of traffic and underdeveloped infrastructure often create conditions for conflicts and dangerous situations on the road. Drivers who violate the rules can cause not only negative consequences for themselves, but also become the cause of serious traffic accidents, which are often accompanied by human injuries and losses.

The study and classification of the behavior of drivers on the road in Ukraine becomes a key stage in understanding this problem and developing effective strategies for its solution. Analysis and understanding of the factors that influence driver behavior can contribute to the development of preventive measures, increase the level of awareness of drivers and improve road safety systems, thereby ensuring greater safety for road users in Ukraine.

The purpose of the study is the development of a software product for the classification of driver behavior on the road using mobile phone sensors.

The object of the study is the process of classifying the driver's behavior on the road and forming a trip assessment.

The subject of the study is a system that collects data from mobile phone sensors during a trip, classifies the driver's behavior on the road, based on the collected data, and forms an assessment of the trip according to the specified criteria for the convenience of the user.

Research methods. To develop the system, technologies that allow reading data from mobile phone sensors, working with maps and performing machine learning that solves the classification problem were used: the Dart programming language, the Flutter framework, the Google Maps API, the ml_algo library, the SQLite DBMS, and the Android Studio development environment .

The practical significance lies in the possibility of using the system during real trips by vehicles, which will allow collecting data from the phone's sensors about the nature of traffic and determining the driver's behavior on the road, providing him with a detailed assessment of the trip. This application of the system will give the user the opportunity to analyze his driving mistakes and improve his skills, which will contribute to reducing the number of road accidents.

Keywords: Bloc, Cubit, layered architecture, sensors, state management, machine learning, classification algorithms, decision tree.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ СКОРОЧЕНЬ І ТЕРМІНІВ 10
ВСТУП 11
1 ОГЛЯД ІСНУЮЧИХ РІШЕНЬ 13
1.1 Постановка завдання 13
1.2 Порівняння аналогів 14
1.2.1 Застосунок Waze..... 15
1.2.2 Застосунок Smooth Driver Monitoring and Mapping…..... 17
1.2.3 Застосунок DriveScore…..... 18
Висновки до розділу 1 22
2 АПАРАТ ВИРІШЕННЯ ПОСТАВЛЕНОЇ ЗАДАЧІ 23
2.1 Опис програмних засобів..... 23
2.1.1 Мова програмування Dart…..... 23
2.1.2 Фреймворк Flutter……..... 25
2.1.3 Середовище розробки Android Studio..... 27
2.2 Огляд архітектури системи…..... 34
2.2.1 Багатошарова архітектура..... 34
2.2.2 Архітектури управління станом системи..... 35
2.2.3 Архітектура BLoC..... 36
2.3 Інтегроване програмне забезпечення..... 48
2.3.1 Google Maps API…….....48
2.3.2 Плагін Geocoding …….....49
2.3.3 Плагін Geolocator..... 50
2.4 Технології для вирішення завдання класифікації поведінки водія..... 53
2.4.1 Плагін sensors_plus..... 53
2.4.2 Плагін ml_algo..... 55
2.4.3 Алгоритм «Дерево рішень»..... 58

Висновки до розділу 2 61

3 РЕАЛІЗАЦІЯ СИСТЕМИ 62

3.1 Загальна структура системи 62

3.2 Набір необхідних даних для класифікації поведінки водія..... 63

3.3 Алгоритми застосовані для класифікації поведінки водія..... 69

3.4 База даних системи……..... 80

3.5 Діаграма екранів User Flow…..... 84

Висновки до розділу 3 87

4 ВЗАЄМОДІЯ КОРИСТУВАЧА З ПРОГРАМНОЮ СИСТЕМОЮ..... 88

4.1 Встановлення системи…….... 88

4.2 Огляд можливостей системи..... 88

Висновки до розділу 4 101

5 РОЗРОБКА СТАРТАПУ ПРОЄКТУ 102

5.1 Опис ідеї проєкту 102

5.2 Технологічний аудит ідеї проєкту 104

5.3 Аналіз ринкових можливостей запуску стартап проєкту 105

Висновки до розділу 5 108

ВИСНОВКИ 112

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 113

ДОДАТОК А 114

ДОДАТОК Б 123

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

ДТП – Дорожньо-транспортна Пригода;

BLoC – Business Logic Component;

MVP – Model View Presenter;

MVVM – Model-View-ViewModel;

API – Application Programming Interface;

СУБД – Система Управління Базами Даних;

SQL – Structured Query Language;

SDK – Software Development Kit;

ML – Machine Learning;

GPS – Global Positioning System;

GUI – Graphical User Interface;

URL – Uniform Resource Locator;

URI – Uniform Resource Identifier;

HTTP – HyperText Transfer Protocol;

JSON – JavaScript Object Notation.

ВСТУП

У сучасному інформаційному суспільстві, де технології активно змінюють усі сфери життя, безпека та ефективність дорожнього руху стають насущною проблемою. Зростаючий обсяг транспортного потоку та постійні зміни у міському середовищі ставлять перед нами завдання вдосконалення систем управління дорожнім рухом. У цьому контексті велике значення набуває вивчення та аналіз поведінки водіїв на дорозі, що може служити основою для розробки програмних застосунків, спрямованих на класифікацію цієї поведінки.

Дана магістерська робота спрямована на вирішення актуальних завдань безпеки та оптимізації транспортного руху на дорогах за допомогою збору даних про поїздки та класифікації поведінки водіїв. Застосування новітніх технологій та алгоритмів машинного навчання у сфері транспортних систем відкриває можливості для створення програм, які здатні аналізувати та класифікувати поведінку водіїв на дорозі.

Метою роботи є розробка та реалізація програмного застосунку, який здатен автоматично визначати та класифікувати різноманітні аспекти поведінки водіїв, такі як дотримання правил дорожнього руху, ступінь агресивності, ефективність використання технічних засобів безпеки та інші. Розроблений програмний продукт має на меті вдосконалити системи моніторингу та управління дорожнім рухом, а також забезпечити підґрунтя для подальших досліджень у цій області.

Проведення комплексного аналізу поведінки водіїв з використанням програмного застосунку може надати цінні дані для розуміння та прогнозування дорожньо-транспортних ситуацій. Такий підхід може виявитися не лише ефективним інструментом для підвищення безпеки на дорогах, але й відкрити нові можливості для розвитку інтелектуальних систем транспортного управління в місті.

Для реалізації системи було обрано фрейморк Flutter, що дозволяє створювати мобільні додатки та містить багато гнучких інструментів, зручних у

використанні, що економить час розробки. Для вирішення проблеми класифікації поведінки водія було застосовано бібліотеку ml_algo, що надає багато можливостей для застосування машинного навчання. В результаті було розроблено системи, що аналізує дані з сенсорів мобільного телефону під час руху транспортного засобу, класифікує поведінку водія, надає оцінку поїздці та містить деталі про рух.

Розробка та реалізація програмного застосунку на базі фрейморку Flutter відкриває нові перспективи для мобільних додатків у сфері транспортних систем. Flutter дозволяє створювати кросплатформенні додатки з однаковим дизайном та функціоналом для різних операційних систем, що робить його ідеальним вибором для покращення зручності користувачів та масштабування проєкту. Застосування цього фрейморку в поєднанні з бібліотекою ml_algo сприяло створенню потужного інструменту для аналізу поведінки водіїв на дорозі.

У контексті подальшого розвитку та вдосконалення систем транспортного управління в місті, розроблений програмний продукт може слугувати основою для інтеграції інтелектуальних рішень та автоматизації. Його здатність аналізувати та класифікувати різноманітні аспекти поведінки водіїв створює можливість вдосконалення систем безпеки та оптимізації руху на дорогах, забезпечуючи більш ефективне використання транспортних мереж.

1 ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

1.1 Постановка завдання

Об’єкт дослідження: реєстрація, розпізнавання та класифікація поведінки водія на дорозі.

Предмет дослідження: програмний застосунок класифікації поведінки водія на дорозі.

Мета роботи: зменшення кількості та наслідків дорожньо-транспортних пригод за допомогою розробки програмного продукту класифікації поведінки водія на дорозі за допомогою використання сенсорів мобільного телефону.

Для досягнення мети треба розв’язати наступні завдання:

- дослідити методології для вирішення проблеми класифікації поведінки водія;
- проаналізувати технології необхідні для реалізації системи;
- спроектувати архітектуру системи;
- розробити програмний застосунок за визначеними методологіями.

В результаті розв’язання цих задач маємо отримати інноваційний і ефективний інструмент, який допоможе водіям розуміти та поліпшувати їхні навички водіння, який включає в себе використання сенсорів мобільного телефону, розробку алгоритму класифікації стилів водіння, функціональність картографування та геолокації, зручний інтерфейс, захист конфіденційності даних, масштабованість, підтримку та отримання зворотного зв'язку від користувачів.

До розробленої системи висуваються наступні вимоги:

- система повинна бути розроблена для мобільних платформ (Android та iOS) та забезпечувати оптимальну продуктивність на різних пристроях;
- використання сучасних технологій та стандартів розробки для забезпечення актуальності та стійкості програмного продукту;

- масштабованість та адаптивність, що означає здатність працювати на різних версіях операційних систем та враховувати можливі зміни в специфікаціях мобільних платформ;
- масштабованість системи для ефективної роботи як з невеликим обсягом даних, так і з великими потоками інформації;
- проведення всебічних тестів для визначення якості та стабільності роботи програмного продукту під різними умовами використання;
- розробка інтуїтивного та зручного для користувача інтерфейсу з елементами взаємодії, що полегшують використання програмного продукту.

1.2 Порівняння аналогів

Перед безпосередньо розробкою програмного забезпечення, що буде вирішувати проблему класифікації поведінки водія на дорозі було досліджено аналогічні системи, так як це дає розуміння про їх переваги та недоліки і взагалі чи є розробка такої системи доцільною. Також це показує ситуацію на ринку, наскільки схожі системи є популярними і чи зможе розроблена система бути цікавою користувачам та мати перспективи розвитку.

Внаслідок проведеного дослідження виявлено, що багато існуючих систем часто обмежуються базовим функціоналом та не завжди приділяють достатньо уваги контекстуальним реаліям водійської поведінки, яка може значно відрізнятися в різних регіонах. Зокрема, нижче надано опис додатків Waze, Smooth Driver Monitoring and Mapping та DriveScore, їх можливостей та недоліків, що було враховано при розробці даної системи. Також у розробленому застосунку приділяється особлива увага адаптації до місцевих особливостей та умов дорожнього руху, щоб забезпечити ефективну та безпечну взаємодію з користувачами в українських умовах. Такий підхід покликаний зробити систему

більш привабливою для користувачів та враховувати їхні потреби в контексті їх реального досвіду водіння.

1.2.1 Застосунок Waze

Waze - це популярний мобільний застосунок для навігації та дорожньої інформації, який відрізняється своєю інтерактивністю та спільнотним підходом до вирішення транспортних завдань. Створений компанією Waze Mobile, застосунок був запущений у 2008 році та швидко завоював популярність завдяки своїй інноваційності та ефективності.

Однією з головних особливостей Waze, що зображено на рисунку 1.1, є можливість користувачів обмінюватися реальними часовими даними про дорожні умови. Користувачі мають можливість надсилати відгуки про затори, аварії, об'єкти на дорозі, що дозволяє системі надавати актуальну та точну інформацію про транспортні події. Крім того, алгоритми застосунку використовують ці дані для побудови оптимальних маршрутів та пропозицій щодо зміни маршруту для уникнення заторів та інших труднощів на дорозі. Waze також враховує інші фактори, такі як швидкість руху, погодні умови, та навіть перепади висоти місцевості, для покращення точності інструкцій та передбачення часу прибуття.

Важливим елементом Waze є його соціальна складова. Користувачі можуть побачити місце розташування своїх друзів на мапі та навіть планувати зустрічі, використовуючи вбудовані функції.

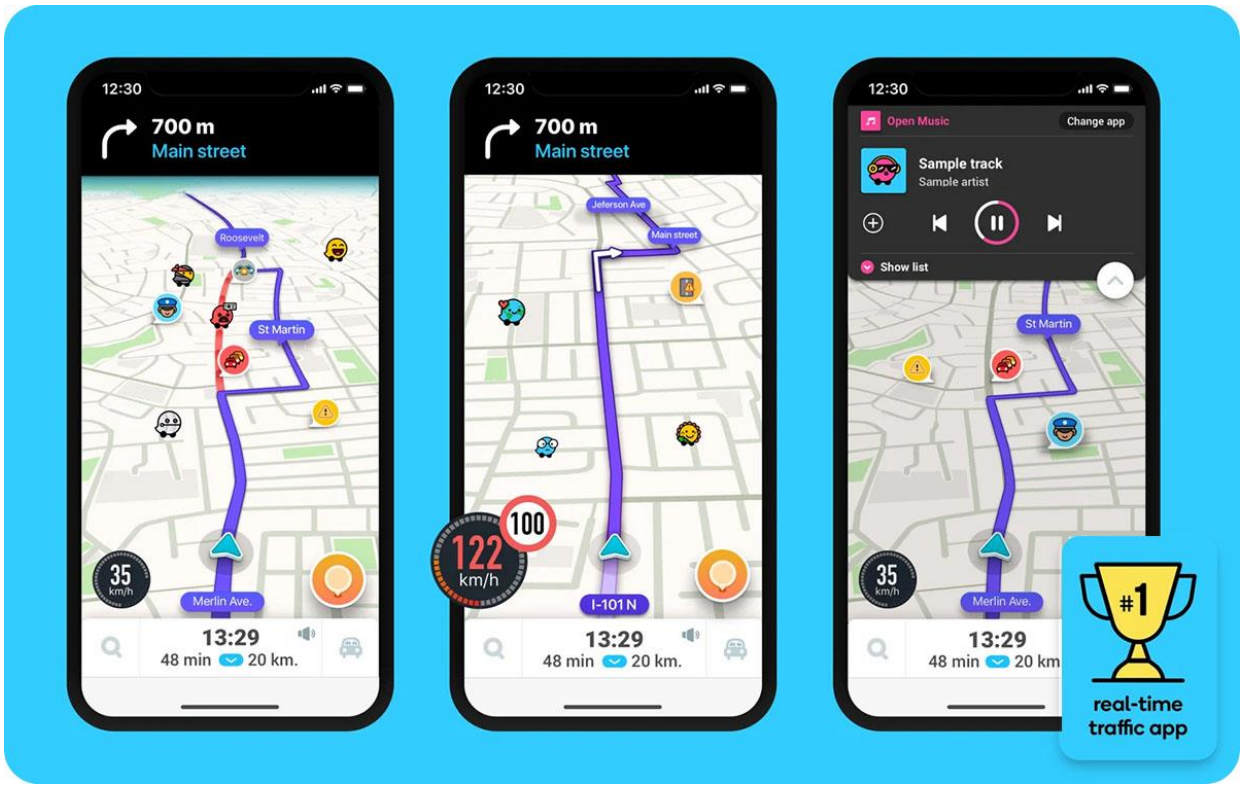


Рисунок 1.1 – Інтерфейс застосунку Waze

Узагальнюючи, Waze визначається як інтелектуальний та соціально-орієнтований застосунок для навігації, що активно використовує дані спільноти для покращення якості та ефективності дорожнього руху.

1.2.2 Застосунок Smooth Driver Monitoring and Mapping

Smooth Driver Monitoring and Mapping – це безкоштовний мобільний застосунок для Android (рис. 1.2), створений компанією Techno Probic. Він входить до категорії Подорожі та навігація і призначений для інтегрованого моніторингу водіння, зокрема, виявлення різких зупинок та відображення їх на карті у вигляді кластерів місць зупинок. Застосунок є надзвичайно корисним для водіїв, які регулярно подорожують по тих самих маршрутах, таких як щоденні поїздки на

роботу чи перевезення вантажів, надаючи можливість здійснювати аналіз та вдосконалювати власну поведінку за кермом.

Застосунок використовує акселерометр і GPS телефону для виявлення різких зупинок та фіксації їх положення, яке подальше відображається на карті у вигляді маркерів. Окрім цього, програма сприяє покращенню керування автотранспортом, роблячи акцент на уникненні різких зупинок та пропонуючи альтернативні маршрути.

Інтерфейс застосунку інтуїтивно зрозумілий, і більшість користувачів буде зручно використовувати екран моніторингу та карту за замовчуванням, що відображає поточне місцезнаходження на карті та події різких зупинок.

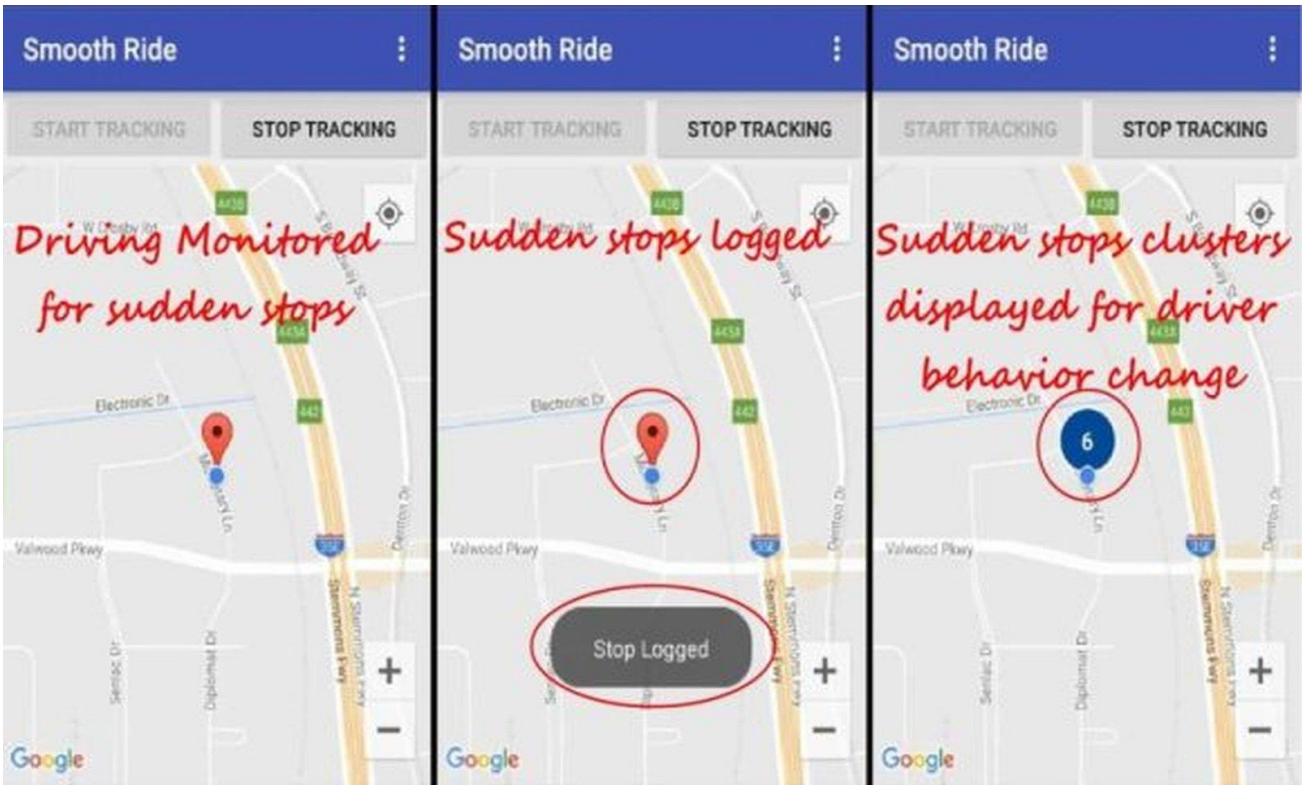


Рисунок 1.2 – Інтерфейс застосунку Smooth Driver Monitoring and Mapping

За необхідності, водії можуть варіювати чутливість виявлення зупинок та очищати історію подій. Застосунок спрямований на зниження кількості різких зупинок, що може призвести до більш ефективного водіння, підвищення рівня безпеки на дорозі, зменшення витрат пального та подовження терміну служби

автомобіля. Smooth Driver Monitoring and Mapping буде корисним як для індивідуальних водіїв, що регулярно подорожують, так і для флотовласників, які прагнуть поліпшити безпеку свого транспортного парку та визначити місцезнаходження зупинок на маршрутах доставки.

1.2.3 Застосунок DriveScore

DriveScore – це нова програма, яка пропонує користувачам довести та вдосконалити свої навички водіння, дозволяючи їм заощадити гроші на цінах автострахування та інших продуктах, що зображено на рисунку 1.3. DriveScore використовує телематичне відстеження через датчики мобільних телефонів для точного збору даних про поведінку водіїв. Ці дані перетворюються на оцінку, яку «досвідчені водії» можуть використовувати для отримання знижок на страхові премії та інші винагороди та заохочення.

Оновлено загальний аудит та дизайн програми, включаючи UX та компоненти. Тепер DriveScore має покращені аналітичні звіти на основі даних для сприяння більшому залученню користувачів і підвищенню цінності програми. Інтегровано заявку на страхування та картки результатів, а також введено керування адміністратором Car Hub для ефективного відстеження та управління технічним обслуговуванням, поновленням дорожнього податку та іншими аспектами.

За допомогою оновлених аналітичних звітів, які надають більше деталей та інсайтів, DriveScore забезпечує користувачів можливістю краще розуміти та вдосконалювати свої навички водіння. Зокрема, користувачі можуть відстежувати свій стиль водіння, оцінювати ефективність використання технічних засобів безпеки та взаємодію з іншими учасниками дорожнього руху. Ця інформація не лише допомагає підвищити рівень безпеки на дорозі, але й визначити конкретні аспекти, які можна покращити для отримання винагород та знижок.

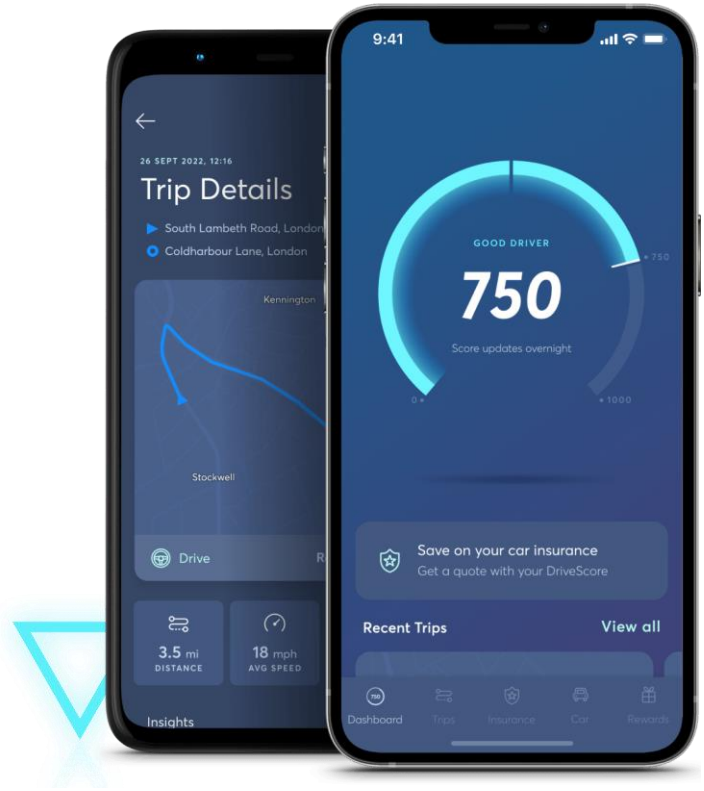


Рисунок 1.3 – Інтерфейс застосунку DriveScore

Пристрій водія використовує машинне навчання для реєстрації всіх поїздок та їх класифікації. Останні 5 поїздок відображаються на інформаційній панелі, яка посилається на повний список подорожей.

Поїздки без автомобіля відображаються як вирізані версії проїзних карток, що можна легко управляти для зручного сканування та визначення неправильно класифікованих поїздок.

Користувач може легко перекласифікувати поїздку, обираючи іншу категорію зі списку класифікації поїздок. Сторінка «Відомості про поїздку» розкриває вичерпну інформацію про ефективність керування автомобілем у конкретній подорожі. Такі «події», як перевищення швидкості чи різке прискорення, відображаються тут і пов’язуються з місцем та часом на карті подорожі. Також відображаються неофіційні дані, такі як погода, місце подорожі, середня швидкість і приблизне споживання палива.

Висновки до розділу 1

В цьому розділі було розглянуто аналогічні системи, які допомагають користувачу відстежувати ситуації на дорогах та сприяють покращенню водіння. Вони є безумовно корисними в застосуванні та надають користувачу багато можливостей, однак і в них є певні недоліки. Зокрема, застосунок Waze не аналізує саму і поїздку, а сконцентрований на аналізі навколишнього середовища. Застосунок Smooth Driver Monitoring and Mapping має доволі обмежений функціонал і дуже простий інтерфейс, а DriveScore не містить всіх важливих деталей в інформації про поїздки.

Як бачимо, на основі проведеного дослідження аналогічних систем, вони можуть містити не весь функціонал, тому для реалізації даного програмного застосунку було виявлено їх недоліки і запропоновано їх вирішення у розроблюваній системі. Зокрема, інтерфейс розроблюваної системи повинен бути інтуїтивно зрозумілим користувачу, інформація про поїздку має містити достатню кількість логів, щоб користувач міг їх аналізувати і покращувати свої навики водіння, що сприятиме покращенню ситуації на дорогах. Також система має містити детальну оцінку поїздки. Крім того, на українському ринку ще немає систем, які вирішують поставлені задачі, тому впровадження такого програмного забезпечення буде сприяти його розвитку.

Крім вищезазначених аспектів, важливою складовою розроблюваної системи є її сумісність з різноманітними платформами та пристроями. Оскільки користувачі мають різні пристрої з різними операційними системами, важливо забезпечити безперебійну роботу програми на різних пристроях. Так як ця система є мобільним додатком, вона була розроблено під платформи Android та IOS. За потреби її можна було б і адаптувати під Web платформу, оскільки обрані технології для її реалізації це дозволяють. Такий підхід розширить коло користувачів та зробить систему більш універсальною в сучасному інформаційному середовищі.

2 АПАРАТ ВИРІШЕННЯ ПОСТАВЛЕНОЇ ЗАДАЧІ

2.1 Опис програмних засобів

Для розробки застосунку був обраний фреймворк Flutter, що використовує мову програмування Dart. Для зберігання даних у системі використовується СУБД SQLite. Програмне забезпечення було розроблено у середовищі Android Studio.

2.1.1 Мова програмування Dart

Dart[1] є мовою програмування загального призначення з відкритим вихідним кодом, впровадженою Google у 2011 році. Вона використовує синтаксис у стилі C та має об'єктно-орієнтований підхід. Основною метою Dart є створення зручних інтерфейсів користувача для веб-сайтів та мобільних додатків. Розробка мови знаходиться на етапі активного розвитку, і вона компілюється до рідного машинного коду для створення швидких мобільних додатків. Dart відзначається впливом інших мов програмування, таких як Java, JavaScript та C#, і володіє жорсткою типізацією.

Мова Dart включає в себе багато загальноприйнятих концепцій програмування, таких як класи, інтерфейси та функції. Важливо відзначити, що Dart не підтримує прямо масиви, але вона використовує збір для реплікації структур даних, таких як масиви, універсали та необов'язкове введення.

Серед переваг Dart можна виділити:

- ефективну продуктивність, що робить його ідеальним для завдань, які вимагають швидкого реагування та надійної роботи. Його легкий для вивчення синтаксис сприяє швидкому освоєнню розробниками, особливо тими, хто вже має досвід роботи з мовами, такими як Java і JavaScript;

- інтеграцію з фреймворком Flutter, що полегшує швидку розробку та оновлення додатків у реальному часі. Крім того, мова підтримує кросплатформену розробку, дозволяючи створювати додатки для різних платформ з єдиної кодової бази;
- сильну типізацію та систему статичного аналізу, які сприяють виявленню помилок на ранніх етапах розробки та підвищують надійність коду. Зростаюча спільнота розробників та розширена екосистема забезпечують підтримку, ресурси та можливості для співпраці та обміну знаннями.

2.1.2 Фреймворк Flutter

Flutter[2] — це безкоштовна платформа мобільного інтерфейсу користувача з відкритим вихідним кодом, створена Google і випущена в травні 2017 року. Загалом вона дозволяє створювати нативну мобільну програму лише з однією кодовою базою. Це означає, що є можливість використовувати одну мову програмування та одну кодову базу для створення двох різних програм (для iOS та Android).

Флаттер складається з двох важливих частин:

1) SDK (набір для розробки програмного забезпечення): набір інструментів, які допоможуть вам розробляти програми. Це включає в себе інструменти для компіляції вашого коду в рідний машинний код (код для iOS і Android);
2) фреймворк (бібліотека інтерфейсу користувача на основі віджетів): набір багаторазово використовуваних елементів інтерфейсу користувача (кнопок, текстових вводів, повзунків тощо), які ви можете персоналізувати для власних потреб.

Крім того, важливо відзначити, що Flutter вирізняється своєю гнучкістю та можливістю розгортати додатки на різних платформах без значного збільшення трудовитрат. Завдяки революційному підходу "одна кодова база", розробники можуть зекономити час та ресурси, створюючи додатки, які функціонують на iOS

та Android. Це особливо важливо в умовах сучасного ринку, де швидкість випуску та наявність додатків на різних платформах можуть визначити успіх продукту.

Ключовою особливістю Flutter є його "гаряче перезавантаження" (Hot Reload), що дозволяє розробникам бачити зміни у реальному часі без необхідності перезапуску застосунку. Цей інструмент значно прискорює процес розробки, дозволяючи в миті вносити зміни, спостерігаючи за їхнім впливом на інтерфейс та функціонал. Гаряче перезавантаження робить Flutter ідеальним для ефективної ітеративної розробки та тестування ідей.

Також важливо зазначити, що Flutter підтримує інтеграцію з різноманітними сторонніми сервісами та API, що робить його високоякісним інструментом для розробки додатків, які взаємодіють з різними платформами та сервісами. Це дає можливість розробникам швидко реалізовувати різноманітні функції та покращувати функціональність своїх додатків, щоб вони відповідали сучасним вимогам та потребам користувачів. Наприклад, в розробленій системі було інтегровано Google Maps Api, що дозволяє відстежувати локацію користувачі та будувати маршрути у застосунку. Також було використано бібліотеку ml_algo для застосування штучного інтелекту для визначення класифікації поведінки водія на дорозі.

2.1.3 СУБД SQLite

SQLite (Structured Query Language) – це вбудована система управління базами даних (СУБД) з відкритим вихідним кодом. Вона відрізняється своєю легкістю використання та простотою встановлення, що робить її популярним вибором для великої кількості проєктів, особливо на мобільних та вбудованих платформах.

Основні характеристики та особливості SQLite:

1) вбудованість: SQLite вбудовується безпосередньо в програму, не потребує окремого сервера або налаштувань. Це полегшує процес розгортання та використання, особливо для додатків з невеликим обсягом даних;

2) легкість використання: синтаксис SQL в SQLite досить простий та стандартизований, що робить його зрозумілим для багатьох розробників. Взаємодія з базою даних може відбуватися через SQL-запити або за допомогою бібліотек для різних мов програмування;

3) локальні файли: база даних SQLite представлена у вигляді одного локального файлу, що спрощує її зберігання та обмін між пристроями. Це особливо корисно для мобільних додатків, які не потребують централізованої серверної архітектури;

4) підтримка транзакцій: SQLite підтримує транзакції, що робить його надійним вибором для операцій, які вимагають атомарності та консистентності даних;

5) невеликий розмір: розмір виконуваного файлу SQLite невеликий, що робить його ідеальним для вбудованих систем та обмежених ресурсів пристроїв;

6) підтримка мов програмування: SQLite має багатомовну підтримку, існують бібліотеки для взаємодії з ним для багатьох популярних мов програмування, таких як Python, Java, C#, і багато інших.

SQLite використовується у різних сценаріях, включаючи роботу з базами даних на мобільних пристроях, вбудованих системах, а також для розробки прототипів та тестування додатків. Так як ця база даних є легкою і швидкою в роботі та дуже добре підходить для розробки мобільних додатків її і було обрано в якості СУБД для розробленої системи.

2.1.4 Середовище розробки Android Studio

Android Studio – це інтегроване середовище розробки (IDE) для створення Android-додатків, розроблене компанією Google. Це потужне інструментаріум, який надає розробникам зручність та ефективність у всіх етапах процесу розробки мобільних додатків для платформи Android.

Основні особливості та характеристики Android Studio:

- Графічний редактор інтерфейсу користувача: Android Studio має вбудований графічний редактор інтерфейсу користувача, що спрощує створення та візуалізацію елементів інтерфейсу додатків.
- Ефективне управління проєктами: Інтелектуальний менеджер проєктів дозволяє легко додавати, видаляти та конфігурувати компоненти застосунку, а також ефективно взаємодіяти з бібліотеками.
- Аналізатор використання ресурсів: Для ефективного управління ресурсами застосунку і підвищення його продуктивності, Android Studio включає аналізатор використання ресурсів, який допомагає виявляти можливі оптимізації та витрати ресурсів.
- Вбудована підтримка мов програмування: Android Studio підтримує різні мови програмування, включаючи Java, Kotlin та C++. Це надає розробникам можливість вибору мови, яка найкраще відповідає їхнім потребам та вподобанням.
- Емулятори та реальні пристрої: Інтегрована система емуляції дозволяє випробовувати додатки на різних версіях Android та віртуальних пристроях. Також, Android Studio легко взаємодіє з реальними пристроями для тестування.
- Широкий спектр інструментів для профілювання: Для оптимізації продуктивності та виявлення можливих проблем, Android Studio включає інструменти для профілювання додатків, включаючи аналіз використання CPU та пам'яті.

Android Studio є невід'ємною частиною процесу розробки Android-додатків, забезпечуючи розробникам потужний та зручний інструментарій для створення інноваційних та високоякісних мобільних застосунків, тому саме воно було обрано для розробки системи.

2.2 Огляд архітектури системи

2.2.1 Багатошарова архітектура

Шаблони багаторівневої архітектури (layered architecture) представляють собою структури з n рівнями, де компоненти системи організовані в горизонтальні шари. Це традиційний підхід до розробки програмного забезпечення, спрямований на досягнення його незалежності. У такій архітектурі всі компоненти взаємодіють, але не є взаємозалежними.

На відміну від цього, архітектура має чотири рівні, де кожен рівень встановлює взаємозв'язок між модульністю та компонентами всередині нього, що зображено на рисунку 2.1. Зверху вниз, ці рівні включають:

- рівень презентації: зберігає всі компоненти, пов'язані з презентаційним шаром;
- бізнес-рівень: містить в собі бізнес-логіку системи;
- рівень постійності: використовується для обробки функцій, таких як об'єктно-реляційне відображення;
- рівень бази даних: Забезпечує зберігання всіх даних.

В даному випадку рівні є закритими, що означає, що запит повинен пройти через всі рівні від верхнього до нижнього. Це робиться з двох причин: для групування схожих компонентів разом та забезпечення їх ізоляції.

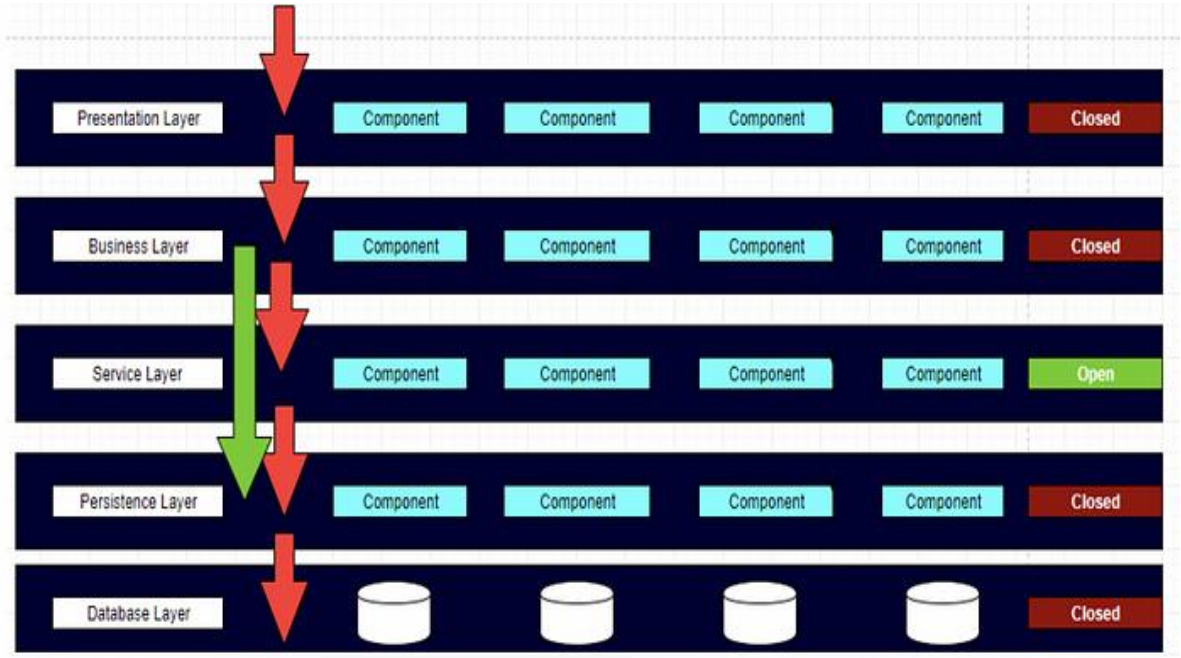


Рисунок 2.1 – Схема багатошарової архітектури

Розглядаючи питання про "подібні" компоненти, важливо враховувати, що всі елементи, пов'язані з конкретним шаром, повинні залишатися в межах цього шару. Це сприяє чіткому розподілу різновидів компонентів та спрощує об'єднання схожого програмного коду на одному рівні. Забезпечуючи ізоляцію шарів, досягається їхня незалежність, що означає, наприклад, що зміна бази даних з сервера Oracle на сервер SQL суттєво вплине лише на рівень бази даних, залишаючи інші рівні непорушеними. Той же принцип працює і у випадку впровадження змін у бізнес-рівень, де зміна механізму бізнес-правил не зачіпає інші рівні.

Шаблон багаторівневої архітектури може бути адаптований для додаткових рівнів, окрім тих, що були згадані. Такий варіант відомий як гібридна багаторівнева архітектура. Наприклад, між бізнес-рівнем і рівнем постійності може бути введений рівень обслуговування. Однак такий сценарій може бути не оптимальним, оскільки вимагає, щоб бізнес-рівень проходив через рівень обслуговування для отримання доступу до рівня постійності. В результаті запити, що проходять через рівень обслуговування, не надають йому жодної додаткової цінності, створюючи тим

самим антишаблон архітектури "воронка". Але цієї проблеми можна уникнути, роблячи рівень обслуговування необов'язковим. Якщо він додає значення запиту, то він використовується; в іншому випадку він обходиться і переходить до відповідного рівня. Це можливо побачити на діаграмі, де запит обходить рівень обслуговування та переходить від бізнес-рівня до рівня постійності.

Варто відзначити, що наявність відкритих шарів може призвести до втрати переваг ізольованих шарів. Зокрема, якщо потрібно замінити рівень постійності, доведеться взяти до уваги не лише рівень відкритих послуг, але і бізнес-рівень. Обидва ці шари будуть зв'язані з рівнем постійності, внаслідок чого важливо розв'язувати проблеми, не шкодячи архітектурі.

Багаторівнева архітектура представляє собою один з найпростіших архітектурних шаблонів у сфері програмного забезпечення. Цей шаблон найбільше виправдовує себе, коли ви маєте намір розробити базову програму з обмеженою кількістю користувачів (менше 100-200) і плануєте, що зміни будуть мінімальними після введення програми в експлуатацію. Реалізація цього шаблону архітектури є найбільш вартісно-ефективною порівняно з іншими архітектурними варіантами.

Нижче відзначені переваги та недоліки багаторівневої архітектури.

Позитивні аспекти:

- легкість перевірки, оскільки компоненти чітко віднесені до визначених шарів і можуть бути перевірені індивідуально;
- простота та легкість реалізації, оскільки більшість програм працюють за принципом розподілу на шари.

Негативні аспекти:

- ускладнення внесення змін в конкретний шар через об'єднану структуру програми, і зчеплення між шарами може ускладнити цей процес, утруднюючи масштабування;
- розгортання в системі вимагає повторне розгортання всієї програми при внесенні змін на певному рівні;

- збільшення розміру програми призводить до збільшення ресурсів, необхідних для проходження запитів через кілька рівнів, що може викликати проблеми з продуктивністю.

Таким чином, необхідно зважити на переваги та недоліки перед вибором багаторівневої архітектури в залежності від специфіки вашого проєкту.

2.2.2 Архітектури управління станом системи

State management в Flutter – це процес управління станом (даними) у програмному застосунку. У Flutter існує кілька підходів до управління станом, і вибір конкретного методу може залежати від розміру вашого застосунку, його складності та інших факторів. Основна ідея – забезпечити ефективний та структурований спосіб взаємодії з даними та їх оновленням відповідно до потреб застосунку.

Основні підходи до управління станом в Flutter:

1) setState. Це базовий метод, вбудований у Flutter. Використовується для оновлення інтерфейсу користувача при зміні стану. Зручно для невеликих додатків та випадків, де стан обмежений.

2) Provider. Пакет, який пропонує спрощений та ефективний механізм управління станом. Забезпечує шлях для доступу до даних з будь-якої частини застосунку. Часто використовується для середнього та великого розміру додатків

3) Bloc (Business Logic Component). Підхід, який розділяє бізнес-логіку від інтерфейсу користувача. Використовується для управління станом та обробки подій. Ефективний для великих та складних додатків.

4) GetX. Легкий та продуктивний пакет для управління станом та навігацією в Flutter. Забезпечує простий API для роботи зі станом та іншими аспектами застосунку. Відмінно підходить для різних розмірів додатків.

5) Redux. Реалізація патерна Flux у Flutter. Забезпечує єдиний нерухомий стан застосунку та передбачає використання actions та reducers для його оновлення. Доцільно для великих додатків зі складним станом.

Вибір конкретного методу залежить від ваших вимог, обсягу та складності проєкту. Кожен підхід має свої переваги та недоліки, іноді комбінування різних методів може бути оптимальним рішенням.

2.2.3 Архітектура BLoC

Bloc (Business Logic Component)[3] – це підхід до управління станом в Flutter, що базується на розділенні бізнес-логіки та інтерфейсу користувача. Bloc допомагає створити відокремлену та тестовану бізнес-логіку, забезпечуючи чистий та ефективний код.

Нижче представлені основні концепції та складові Bloc.

1. Блок (Bloc) – це клас, що містить бізнес-логіку та управляє станом. Блок взаємодіє з подіями та видає стани.

2. Події (Events) – це дії чи події, що відбуваються у застосунку, такі як натискання кнопок, завантаження даних тощо. Блок приймає події та видає стани відповідно до них.

3. Стани (States) – це представлення стану застосунку в конкретний момент часу. Зміни станів спричиняють перебудову інтерфейсу користувача.

4. Потік подій та станів (Event and State Streams). Події та стани передаються між компонентами застосунку за допомогою потоків (streams). Це забезпечує асинхронну обробку подій та оновлення стану.

5. Інтерфейс користувача (UI Layer) відповідає за відображення станів та взаємодію з користувачем. Опрацьовує події та відображає відповідні стани.

Переваги використання Bloc управлінням станом у Flutter:

- розділення бізнес-логіки: відокремлення бізнес-логіки від інтерфейсу користувача допомагає підтримувати чистоту та структурованість коду;
- тестованість: бізнес-логіка може бути легко тестована, оскільки вона не залежить від конкретного інтерфейсу користувача;
- реактивність: використання потоків дозволяє асинхронно обробляти події та оновлювати інтерфейс користувача.

Однак важливо враховувати, що використання Bloc може виявитися зайвим для менших додатків, але для великих проєктів з складною бізнес-логікою цей підхід може бути дуже корисним.

Bloc та Cubit є важливими концепціями в архітектурі Flutter для управління станом. Однак Cubit є спрощеною версією Bloc та використовується в ситуаціях, коли гнучкість та простота є більш пріоритетними, ніж повна потужність Bloc.

Bloc включає в себе бізнес-логіку та відповідає за управління станом. Це абстракція, яка приймає події (Events) та видає стани (States) відповідно до цих подій. Використовується разом із Streams для асинхронного обміну даними між компонентами застосунку.

Cubit — це абревіатура від "Cube" і "Bloc", і він є спрощеною версією Bloc.

Cubit призначений для ситуацій, де потрібно менше бойових магістралей та більше простоти. Має тільки метод emit, який видає новий стан відповідно до подій. Забезпечує більш легку та менш складну реалізацію порівняно з повноцінним Bloc.

Основні принципи Cubit:

- стан (State). Cubit видає стани для відображення в інтерфейсі користувача. Повертає незмінювані об'єкти стану;
- події (Events). Події є тригерами, які змінюють стан Cubit. Викликають метод emit, щоб видали новий стан;
- метод emit. Викликається для оновлення стану. Приймає новий об'єкт стану та сповіщає всіх слухачів;
- відсутність потоків. Не використовує потоки та обіймає простоту.

Переваги Cubit:

- простота. Cubit має менше вагомих концепцій і менше бойових магістралей, що дозволяє розробникам швидше реалізовувати бізнес-логіку.
- менше рутинності. Забезпечує простоту без втрати частини функціональності.

Недоліки Cubit: обмежені можливості. Cubit може бути недостатнім у складних сценаріях, де потрібна велика кількість станів та подій.

Застосування Cubit:

- прості додатки. Cubit є ідеальним вибором для простих або менш розгалужених додатків:
- прототипування. Використовується для швидкого створення прототипів та тестового застосунку.

Архітектура Flutter з використанням Bloc базується на розділенні бізнес-логіки, інтерфейсу користувача та управління станом як показано на рисунку 2.2. Bloc (Business Logic Component) дозволяє вам визначити ці компоненти окремо, забезпечуючи чистоту коду та легше тестування.

Основні компоненти архітектури з використанням Bloc:

1) UI Layer (Шар інтерфейсу користувача):

- віджети: Відповідають за представлення та рендеринг інтерфейсу користувача;
- BlocBuilder і BlocListener: Використовуються для підключення до Bloc і оновлення UI відповідно до змін стану;

1) Business Logic Layer (Шар бізнес-логіки). Cubit або Bloc: Визначається бізнес-логіка та управління станом. Обробляє події і видає нові стани;

2) Event Layer (Шар подій). Events – це тригери або події, які ініціюють зміни стану у Bloc або Cubit;

3) State Layer (Шар стану). Компонент States представляють різні стани, які може мати застосунок. Кожен стан повідомляє про певний аспект вашого інтерфейсу чи бізнес-логіки;

4) Repository Layer (Шар репозиторіїв). Компонент Repository організовує отримання та зберігання даних. Забезпечує інтерфейс для отримання даних від різних джерел (наприклад, віддалені API або локальна база даних);

5) Data Layer (Шар даних). Цей шар містить:

- Data Models. Представлення даних у застосунку;
- Data Providers. Реалізація отримання та зберігання даних (наприклад, класи для роботи з базою даних або взаємодії з API).

Процес взаємодії:

1) коли відбувається взаємодія користувача з інтерфейсом (наприклад, кнопка натискання), створюється відповідна подія (Event);

2) ця подія передається в Bloc або Cubit, який обробляє її та ініціює зміни стану;

3) Bloc або Cubit видає новий стан;

4) BlocBuilder або BlocListener в UI Layer реагує на зміни стану і оновлює відповідно інтерфейс користувача.

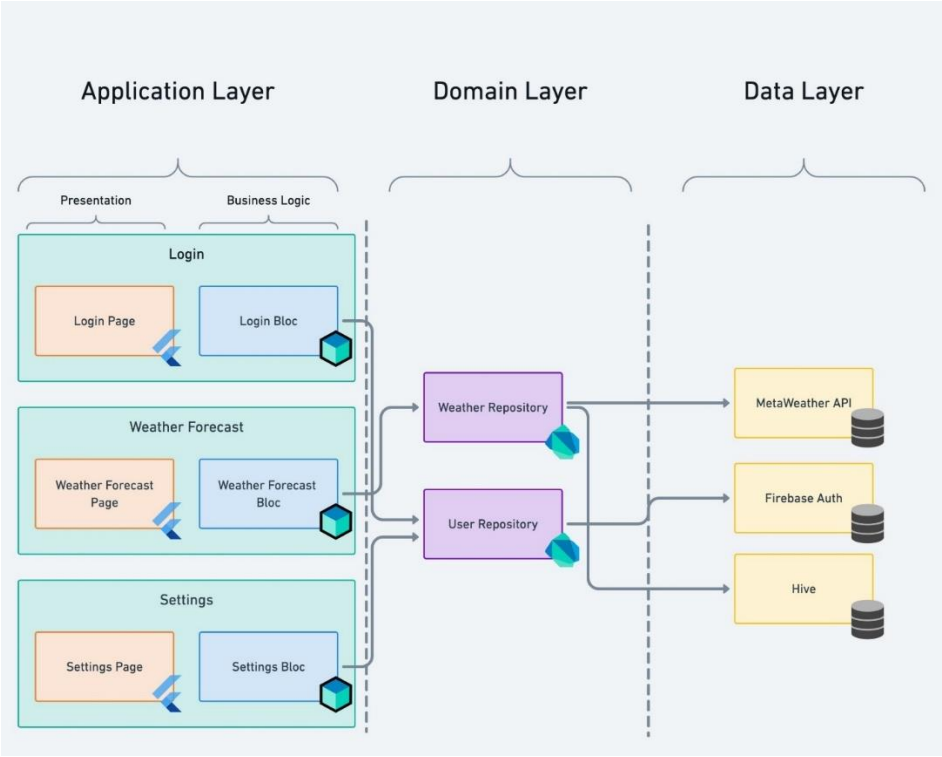


Рисунок 2.2 – Схема BLoC архітектури

Використання Bloc у Flutter дозволяє добре структуровано організувати ваш код, зробити його більш підтримуваним і дозволяє ефективно взаємодіяти з різними частинами застосунку. Такий архітектурний патерн є добре сумісний із підходом багатошарової архітектури саме тому він був обраний для розробки системи.

2.3 Інтегроване програмне забезпечення

2.3.1 Google Maps Api

Google Maps API в Flutter відкриває широкі можливості для інтеграції потужних функцій Google Maps безпосередньо в ваші мобільні додатки. Щоб зручно взаємодіяти з Google Maps API в Flutter, ви можете використовувати пакет google_maps_flutter. Цей пакет надає простий та ефективний інтерфейс для взаємодії з картами Google Maps у застосунку.

Основні функції та можливості пакету google_maps_flutter:

1) відображення карти. Використовуючи цей плагін, можна з легкістю вставляти та відобрати карти Google Maps безпосередньо в інтерфейсі Flutter-застосунку. Це дозволяє користувачам зручно взаємодіяти з картами та отримувати актуальну географічну інформацію;

2) маркери та інфо-вікна. Можна також додавати маркери на карту, позначаючи важливі місця або об'єкти і налаштовуйте інфо-вікна, щоб надавати додаткову інформацію при кліці на маркер. Це робить взаємодію з картами максимально зрозумілою для користувачів;

3) полігони та лінії. Є можливість відображати полігони та лінії на карті, щоб візуально виділити та намічати області чи маршрути. Це особливо корисно для додатків, які пов'язані з відображенням географічних даних;

4) зум та переміщення карти. Інтегроване керування зумом та переміщенням карти дозволяє користувачам максимально комфортно оглядати різні регіони. Зручні жести дозволяють легко маніпулювати рівнем збільшення та переміщенням.

Інтеграція Google Maps API у Flutter через google_maps_flutter відкриває широкі можливості для розширення функціональності вашого застосунку та створення інтерактивних та зручних картографічних рішень.

2.3.2 Плагін Geocoding

Для роботи з геокодуванням у Flutter можна використовувати плагін "geocoding"[4], який дозволяє вам отримувати адресу на основі координат (обернений геокодинг) або координати на основі адреси (геокодування). Цей плагін надає зручний інтерфейс для взаємодії з різними сервісами геокодування, такими як Google Maps, Here, OpenStreetMap і багатьма іншими.

Основні функції та можливості плагіна включають:

1) геокодування (Geocoding): Ця функція дозволяє перетворювати адресу на координати. Ви можете вказати текстову адресу, і плагін надасть вам відповідні географічні координати;
2) обернений геокодинг (Reverse Geocoding): Ця функція дозволяє отримувати адресу на основі координат. Ви подаєте географічні координати, і плагін надає вам відповідну адресу;
3) підтримка різних провайдерів: Плагін може використовувати різні сервіси геокодування, що розширює ваш вибір. Це може бути зручно, особливо якщо ви маєте вимоги до конкретного сервісу;
4) асинхронні операції: Оскільки геокодування може вимагати взаємодії з мережею, всі операції виконуються асинхронно. Це дозволяє уникнути

блокування інтерфейсу користувача і забезпечити більш гладку роботу застосунку.

2.3.3 Плагін Geolocator

У Flutter для роботи з геолокацією, тобто отримання інформації про поточне місцезнаходження пристрою, можна використовувати різноманітні плагіни. Один з популярних плагінів для роботи з геолокацією в Flutter – geolocator.

Geolocator – це Flutter-плагін, який надає можливість використовувати сервіси геолокації для отримання інформації про місцезнаходження пристрою.

Властивості та функціональність:

1) отримання координат: плагін дозволяє отримувати поточні координати (широта і довгота) пристрою;

2) отримання інформації про місцезнаходження: можливість отримати додаткову інформацію про місцезнаходження, таку як швидкість, висота, точність і т. д.;

3) визначення доступності геолокації: перевірка, чи включений сервіс геолокації на пристрої;

4) слухання змін місцезнаходження: можливість підписуватися на зміни місцезнаходження та отримувати їх у реальному часі.

Додатково, важливо зазначити, що використання плагіна geolocator у Flutter не лише дозволяє отримувати статичні дані про місцезнаходження, але й надає можливість інтегрувати динамічну інформацію для поліпшення функціональності додатків. Зокрема, слухання змін місцезнаходження в реальному часі відкриває широкі можливості для розробників у реалізації додаткових функцій, таких як автоматичне оновлення маршрутів або надсилання повідомлень користувачам щодо поточного стану дорожнього руху.

2.4 Технології для вирішення завдання класифікації поведінки водія

2.4.1 Плагін sensors_plus

Плагін для Flutter, що надає доступ до датчиків акселерометра, гіроскопа та магнітометра, отримав позначку sensors_plus[5]. Додавши цей плагін у ваш файл pubspec.yaml, ви відкриваєте доступ до класів подій цих датчиків через потоки.

Класи подій включають:

- UserAccelerometerEvent. Надає інформацію про прискорення пристрою в м/с². Цей клас виявляє прискорення пристрою, враховуючи його рух по прямій лінії та зміну швидкості у різних напрямках. Дані отримані з цього потоку фільтруються для виключення впливу гравітації, і результат подається через AccelerometerEvent;
- AccelerometerEvent. Надає необроблені дані про прискорення пристрою в м/с², включаючи вплив сили тяжіння. Цей потік виводить дані без будь-якої постобробки, що дозволяє отримати необроблені показники з акселерометра;
- GyroscopeEvent. Надає інформацію про обертання пристрою;
- MagnetometerEvent. Описує навколишнє магнітне поле навколо пристрою, що може бути використано, наприклад, для створення компаса.

Ці події доступні через BroadcastStream, і кожен з них має свій відповідний потік: accelerometerEvents, userAccelerometerEvents, gyroscopeEvents і magnetometerEvents.

Плагін sensors_plus був використаний у розробленій системі для отримання динамічної інформації про пристрій, яка включає дані з сенсорів прискорення та обертання, що зчитуються під час поїздки транспортним засобом.

2.4.2 Плагін ml_algo

Мета бібліотеки ml_algo – надати нативну реалізацію алгоритмів машинного навчання для мови Dart для тих, хто зацікавлений у мові Dart та науці про дані. Ця бібліотека спеціально спрямована на використання в Dart VM і Flutter, але вона недоступна для веб-додатків.

Нижче прдеставлені основні можливості та компоненти бібліотеки.

1. Алгоритми класифікації:
    - LogisticRegressor: клас для лінійної двійкової класифікації даних;
    - LogisticRegressor.SGD: реалізація алгоритму логістичної регресії на основі стохастичного градієнтного спуску із регуляризацією L2;
    - LogisticRegressor.BGD: реалізація алгоритму логістичної регресії на основі пакетного градієнтного спуску із регуляризацією L2.
    - LogisticRegressor.newton: реалізація алгоритму логістичної регресії на основі методу Ньютона-Рафсона із регуляризацією L2;
    - SoftmaxRegressor: клас для лінійної багатокласової класифікації даних;
    - DecisionTreeClassifier: клас для класифікації за допомогою дерев рішень;
    - KnnClassifier: клас для класифікації за допомогою алгоритму k найближчих сусідів.
2. Алгоритми регресії:
    - LinearRegressor: загальний клас для пошуку лінійного шаблону в навчальних даних та прогнозування результатів у вигляді дійсних чисел;
    - LinearRegressor.lasso: реалізація алгоритму лінійної регресії із регуляризацією ласо;
    - LinearRegressor.SGD: реалізація алгоритму лінійної регресії на основі стохастичного градієнтного спуску із регуляризацією L2;

- LinearRegressor.BGD: реалізація алгоритму лінійної регресії на основі пакетного градієнтного спуску із регуляризацією L2;
- LinearRegressor.newton: реалізація алгоритму лінійної регресії на основі методу Ньютона-Рафсона із регуляризацією L2;
- KnnRegressor: клас для роботи з навчальними даними та прогнозування для кожного нового спостереження на основі k найближчих сусідів.

3. Алгоритми кластеризації та пошуку:
   - KDTree: Алгоритм для ефективного пошуку даних;
   - Гешування з урахуванням місцевості: Сімейство алгоритмів для ефективного пошуку K найближчих сусідів, розбиваючи дані на бункери.

Також доступний вибір моделі за допомогою використання коспоненту CrossValidator, що являє собою фабрику для створення екземплярів крос-валідаторів. Перехресна перевірка дозволяє дослідникам налаштовувати гіперпараметри алгоритмів машинного навчання для різних частин набору даних.

2.4.3 Алгоритм «Дерево рішень»

Дерево рішень – це високоефективний інструмент в арсеналі алгоритмів машинного навчання, який використовується для завдань класифікації та регресії. Його структура нагадує блок-схему, де кожен внутрішній вузол виконує перевірку атрибута, гілка представляє результат тесту, а кожен кінцевий вузол (листок) містить мітку класу. Дерево рішень створюється шляхом рекурсивного розбиття навчальних даних на підмножини, доки не буде досягнуто критерію зупинки, такого як максимальна глибина дерева чи мінімальна кількість зразків для розбиття вузла.

Під час навчання алгоритм дерева рішень обирає найкращий атрибут для поділу даних, використовуючи метрики, такі як ентропія або домішка Джині, для

вимірювання однорідності підмножин. Мета полягає в знаходженні атрибута, який максимізує приріст інформації або зменшення домішок після поділу.

Дерево рішень – це структура, подібна до блок-схеми, де кожен внутрішній вузол представляє функцію, гілки вказують правила, а листки містять результати алгоритму як показано на рисунку 2.3. Використовується для класифікації та регресії, а також у складі Random Forest[6] для навчання на різних підмножинах даних, роблячи його одним із потужних алгоритмів машинного навчання.

Термінологія дерева рішень:

- Кореневий вузол Найвищий вузол у дереві, представляє повний набір даних.
- Рішення/внутрішній вузол: Вузол, що вибирає вхідну функцію.
- Листовий/кінцевий вузол: Вузол, який містить мітку класу або числове значення.
- Поділ: Процес розділення вузла на підвузли за допомогою критерію поділу та вибраної функції.
- Гілка/піддерево: Частина дерева, що починається у внутрішньому вузлі та закінчується листками.
- Батьківський вузол: Вузол, що розділяється на дочірні вузли.
- Дочірній вузол: Вузли, що виникають при розділенні батьківського вузла.
- Домішка: Міра однорідності цільової змінної в підмножині даних.
- Дисперсія: Міра того, наскільки прогнозовані та цільові змінні відрізняються в різних вибірках даних.
- Інформаційний приріст: Міра зменшення домішок через поділ набору даних за допомогою конкретної функції.
- Обрізка: Процес видалення гілок, які не додають інформації, щоб уникнути надмірної складності.

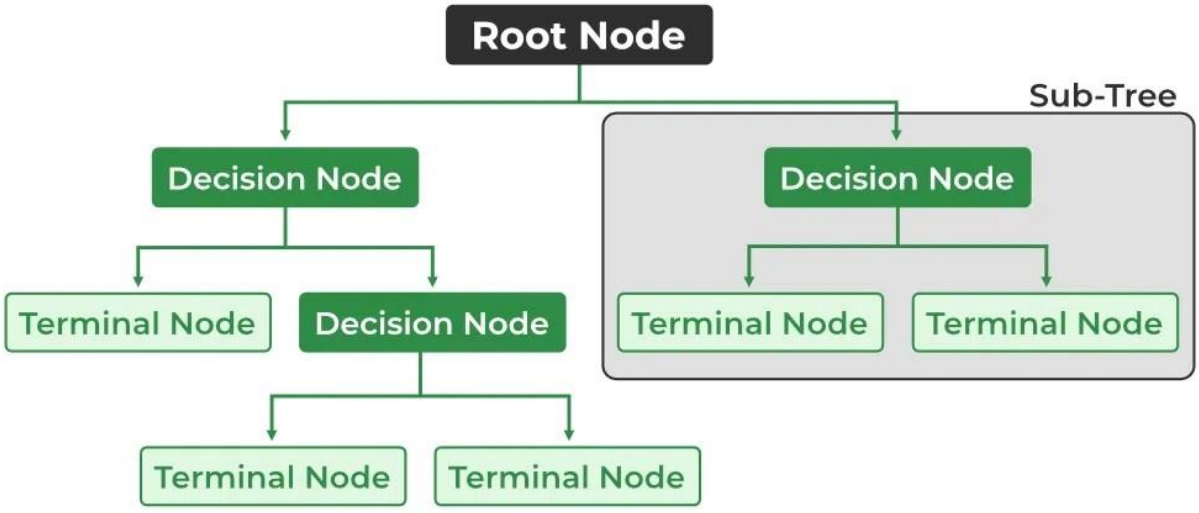


Рисунок 2.3 – Схема алгоритму дерева рішень

Процес побудови дерева рішень включає в себе етап вибору атрибутів, який грає важливу роль у формуванні структури дерева. Основні кроки цього етапу розглядаються через призму міри вибору атрибутів (ASM), яка визначає корисність різних атрибутів для розділу набору даних.

Побудова дерева рішень: дерево рішень навчається шляхом рекурсивного розділення вихідного набору на підмножини з використанням міри ASM. Мета ASM – визначити атрибут, який максимізує однорідність підмножин після поділу, і тим самим максимізує приріст інформації. Процес, відомий як рекурсивне розбиття, повторюється для кожної підмножини до завершення у випадку однорідності цільової змінної або неможливості подальшого розділення.

Для класифікатора дерева рішень не потрібні специфічні знання предметної області або тонка настройка параметрів, що робить його відмінним для дослідницького виявлення знань. Враховуючи обробку великих обсягів даних, дерева рішень забезпечують ефективність в опрацюванні наборів великих розмірів.

Ентропія служить мірою ступеня випадковості або невизначеності в наборі даних. У випадку класифікацій, ентропія визначається розподілом міток класів у

вихідному наборі. Для підмножини даних у вузлі дерева ентропія визначається формулою (2.1):

$$Hi = -\sum k \epsilon K np(i,k) log2p(i,k), \quad (2.1)$$

де p(k) — частка точок даних,

k — клас.

Важливі моменти, пов'язані з ентропією, включають те, що вона дорівнює 0 при повній однорідності і максимізується при рівномірному розподілі класів, що вказує на максимальну невизначеність.

Ентропія використовується для оцінки якості поділу, спрямовуючи вибір атрибута, що мінімізує ентропію в результуючих підмножинах. Критерій поділу вибирає атрибут з найвищим приростом інформації, спрямовуючи рекурсивний процес побудови дерева рішень.

Домішка Джіні виступає як критерій, що визначає точність розподілу між класифікованими групами в наборі даних. Оцінка домішки Джіні знаходиться в межах від 0 до 1, де 0 вказує на повну однорідність, а 1 – на випадковий розподіл елементів. Ми прагнемо досягти якнайменшого значення індексу Джіні. Цей індекс розраховується за формулою (2.2):

$$Gini\ Impurity = 1 - \sum pi2, \quad (2.2)$$

де pi — частка елементів у множині,

i — категорія.

Приріст інформації визначає зменшення ентропії або дисперсії через розділення набору даних за певною властивістю. Цей показник використовується в алгоритмах дерева рішень для оцінки корисності функції за рахунок формування більш однорідних підмножин з огляду на мітки класу чи цільову змінну. Чим вищий приріст інформації, тим цінніше функція для прогнозування цільової змінної.

Інформаційний приріст атрибута A відносно набору даних S обчислюється за формулою (2.3):

$$Information\ Gain(H,\ A) = H - \sum HHV, \quad (2.3)$$

де A — конкретний атрибут або мітка класу,

|H| — ентропія зразка набору даних S,

|HV| – кількість екземплярів у підмножині S.

Приріст інформації вимірює зменшення ентропії або дисперсії, досягнуте поділом набору даних за атрибутом A. Атрибут, який максимізує приріст інформації, вибирається як критерій поділу для побудови дерева рішень. Збільшення інформації застосовується як у класифікації, так і в деревах рішень регресії, пристосовуючись до особливостей конкретного випадку.

Алгоритм дерева рішень використовується для класифікації набору даних шляхом аналізу та прогнозування його класів. Починаючи з кореневого вузла, алгоритм порівнює значення кореневого атрибута з атрибутом запису в наборі даних і рухається гілкою до наступного вузла.

Цей процес повторюється для кожного наступного вузла, порівнюючи значення атрибутів і продовжуючи процес далі. Досягається вузол листа, коли не можна далі класифікувати вузли, і остаточний вузол стає листовим вузлом. Алгоритм можна розглядати через наступним чином:

- крок 1: початок дерева в кореневому вузлі S з повним набором даних;
- крок 2: знаходження найкращого атрибуту у наборі даних за допомогою вимірювання вибору атрибутів (ASM);
- крок 3: розділення S на підмножини з можливими значеннями для найкращих атрибутів;
- крок 4: створення вузла дерева рішень, який містить найкращий атрибут;
- крок 5: рекурсивне створення нових дерев рішень, використовуючи підмножини набору даних, створені на кроці 3. Продовження цього процесу до досягнення стадії, коли вузли більше не можна класифікувати, і остаточний вузол стає листовим вузлом.

Переваги дерева рішень:

- простота в зрозумінні, слідуючи природньому процесу прийняття рішень людиною;

- велика корисність для вирішення проблем прийняття рішень;
- допомагає розглянути всі можливі наслідки проблеми;
- вимагає менше очищення даних у порівнянні з іншими алгоритмами.

Недоліки дерева рішень:

- має багато шарів, що робить його складним. Є можливість перенавчання, яке можна вирішити за допомогою алгоритму випадкового лісу;
- зі збільшенням кількості міток класу обчислювальна складність може зростати.

Проблеми, які підходять для аналізу дерева рішень, можна класифікувати за декількома ключовими критеріями. Незважаючи на те, що існують різні методи навчання дерев рішень, кожна проблема має свої унікальні особливості. Ось деякі з характеристик, які роблять проблему ідеальною для вивчення дерева рішень:

- представлення екземплярів. Використання пар атрибут-значення для опису екземплярів є ключовим елементом вивчення дерева рішень. Це стосується визначення атрибутів, таких як температура, та їхніх відповідних значень, таких як "гаряче". Важливо, щоб атрибути мали чітко визначені значення, спрощуючи побудову структури дерева рішень;
- дискретні вихідні значення цільової функції. Ефективність дерева рішень зумовлена чітко визначеними результатами цільової функції. Зазвичай дерева рішень застосовуються для задач категоризації, де вихідні значення є дискретними, наприклад, "так" чи "ні". Така структура легко розширюється для вирішення проблем із подвійними мислимими значеннями результатів;
- розділові описи. Дерева рішень ефективно використовують диз'юнктивні вирази для представлення різних розділових описів. Це дозволяє враховувати різноманітність в навчальних даних та створює гнучку структуру для вирішення різних сценаріїв;
- стійкість до помилок у навчальних даних. Дерева рішень володіють високою стійкістю до помилок у навчальних даних, включаючи невідповідності та

розбіжності в категоризації зразків. Це робить їх ефективними для вирішення завдань із навчальними даними різної якості;

- можливість роботи з відсутніми значеннями. Врахування відсутніх значень атрибутів у навчальних даних є важливим аспектом. Дерева рішень можуть ефективно працювати з такими випадками, що робить їх відмінними для використання у реальних умовах, де не завжди можна отримати повну інформацію.

Для створення дерева рішень застосовується алгоритм CART (Classification and Regression Trees). Цей алгоритм вибирає оптимальний розподіл у кожному вузлі, базуючись на показниках, таких як домішка Джині або приріст інформації. Основні етапи алгоритму CART включають наступне:

- повний набір даних у кореневому вузлі. Кореневий вузол дерева повинен містити весь набір навчальних даних;
- визначення домішок даних для кожної ознаки. Оцінюються домішки даних на основі кожної ознаки в наборі даних. Це може бути виміряно за допомогою показників, таких як індекс Джині або ентропія для класифікації та різні метрики для регресії;
- вибір функції для розділення даних. Обирається функція, яка забезпечує найбільший приріст інформації або мінімізацію домішок при розділенні даних;
- розділення даних на підмножини. Для кожного значення вибраної функції розділіть набір даних на дві підмножини – одну, де об'єкти мають обране значення, і іншу, де вони його не мають. Розділення повинно створити підмножини, які є максимально чистими відносно цільової змінної;
- визначення домішок для отриманих підмножин. На основі цільової змінної визначаються домішки для кожної отриманої підмножини;
- повторення процесу для кожної підмножини. Кроки 2–5 повторюються для кожної підмножини, поки не буде виконана умова зупинки. Це може бути обумовлено максимальною глибиною дерева, мінімальною кількістю зразків для розділення або мінімальною домішкою;

- призначення мітки кінцевого вузла. Кожному листовому вузлу дерева призначається мітка класу більшості у завданнях класифікації або середнє значення у завданнях регресії.

Алгоритм класифікації та регресії[7] для класифікації можна виразити формулою (2.4):

$$G(Qm, tm) = nmnmleftH(Qmleft(tm)) + nmnmrightH(Qmright(tm)), \quad (2.4)$$

де H – міра домішок лівої та правої підмножин у

m – вузол,

nm – кількість екземплярів у лівій і правій підмножинах у вузлі m.

Нижче представлені переваги та недоліки використання методу дерева рішень.

Сильні сторони:

- зрозумілі правила. Дерева рішень генерують правила, які легко інтерпретувати і зрозуміти, що робить їх ефективним інструментом для роз'яснення прийнятих рішень;
- ефективна класифікація. Вони забезпечують високу швидкодіючу класифікацію без значних обчислювальних витрат;
- обробка різних типів змінних. Дерева рішень добре впораються як із безперервними, так із категоріальними змінними, розширюючи їхню універсальність;
- чітка індикація важливих факторів. Модель надає ясні вказівки стосовно важливості полів для прогнозування чи класифікації, сприяючи зрозумінню впливу різних факторів;
- простота використання. Дерева рішень легкі у використанні та не потребують великої технічної експертизи, що робить їх доступними для широкого кола користувачів;
- масштабованість та паралельна обробка. Масштабованість дерев дозволяє їм ефективно обробляти великі обсяги даних та використовувати паралельну обробку для скорочення часу аналізу;

- толерантність до відсутніх значень. Можливість обробляти відсутні значення робить дерева рішень відмінним вибором для даних з відсутніми або неповними записами;
- обробка нелінійних зв'язків. Можливість обробляти нелінійні зв'язки між змінними надає їм ефективність у розв'язанні складних наборів даних;
- обробка незбалансованих даних. Дерева рішень можуть ефективно враховувати незбалансовані дані, враховуючи важливість окремих вузлів на основі розподілу класів.

Слабкі сторони підходу дерева рішень:

- неідеальність для завдань оцінки безперервних атрибутів. Дерева рішень менше підходять для завдань, де основною метою є передбачення значень безперервного атрибута;
- проблеми з класифікацією при великій кількості класів і обмежених прикладах. У випадках класифікації з великою кількістю класів та обмеженим обсягом навчальних прикладів, дерева рішень можуть виявлятися схильними до помилок;
- високі обчислювальні витрати: процес вирощування дерева рішень може бути обчислювально витратним, оскільки потрібно сортувати кожне поле розбиття перед знаходженням оптимального розбиття. Алгоритми також можуть вимагати обчислення оптимальних комбінованих ваг;
- тенденція до перенавчання: дерева рішень схильні до перенавчання, особливо коли вони стають дуже глибокими або складними, що може призвести до низької адаптації до нових, невидимих даних;
- чутливість до змін в навчальних даних: малі зміни в навчальних даних можуть призвести до різних структур дерев рішень, ускладнюючи порівняння та відтворення результатів;

- обмежена обробка відсутніх даних. Багато алгоритмів дерев рішень погано обробляють відсутні дані, що може вимагати імпутації або видалення записів із відсутніми значеннями;
- упередженість через початкові критерії розбиття. Використані початкові критерії розбиття можуть призводити до упереджених дерев, особливо в незбалансованих наборах даних або при рідких класах;
- обмежена здатність моделі представляти складні зв'язки. Дерева рішень мають обмежену здатність виражати складні зв'язки між змінними, особливо у випадках нелінійних чи інтерактивних ефектів;
- чутливість до масштабування вхідних функцій. Масштабування вхідних функцій може впливати на рішення дерев, особливо при використанні метрик на основі відстані або правил прийняття рішень, що залежать від порівняння між значеннями.

Висновки до розділу 2

В даному розділі було описано технології і архітектури, використані системи. В якості мови програмування було обрано мову Dart, яка використовується в контексті фрейморку Flutter. Систему було розроблено в середовищі Android Studio. В якості архітектури було обрано багатошарову архітектуру, так як вона зручно розділяє концепції в програмуванні, що робить код програми зручним для розширень. В якості підходу управління станом було обрано BLoC, оскільки він є добре оптимізованим.

Також було проведено аналіз технологій для вирішення поставлених задач функціональності системи. Зокрема, для роботи з картою було інтегровано в систему Google Maps Api, для роботи з координатами – geocoding плагін, а для роботи з локацією – geolocator плагін. Для зчитування динамічних даних із сенсорів смартфону було обрано sensor_plus плагін, що має доступ до сенсорів обертання,

прискорення та магнітометра, а для вирішення проблеми класифікації поведінки водія було обрано бібліотеку ml_algo, яка дозволяє використовувати алгоритми машинного навчання. Крім того, було проаналізовано алгоритми для завдань класифікації і було визначено, що достатньо хороші результати показує алгоритм «дерево рішень» і було описано його роботу.

Вибір мови програмування Dart та фреймворку Flutter є стратегічним рішенням, оскільки це надає можливість розробляти кросплатформенні мобільні додатки, що працюють як на платформі Android, так і на iOS. Окрім того, використання середовища Android Studio дозволяє комфортно розробляти та тестувати програму, використовуючи широкий спектр інструментів, що підвищує продуктивність та ефективність розробки.

Багатошарова архітектура обрана з метою підвищення гнучкості та чіткого розділення функціональних блоків. Це сприяє кращому управлінню кодом та спрощує його розширення. Обрана архітектура BLoC, або BloC (Business Logic Component), допомагає ефективно управляти станом застосунку, забезпечуючи чітку структуру та швидкодіючий код.

3 РЕАЛІЗАЦІЯ СИСТЕМИ

3.1 Загальна архітектура системи

Як було зазначено у другому розділі для побудови системи було обрано багатошарову архітектуру, яка дозволяє розділяти компоненти по різних рівнях, що робить код добре структурованим та зрозумілим для розробника.

На рисунку 3.1 зображена загальна архітектура системи:

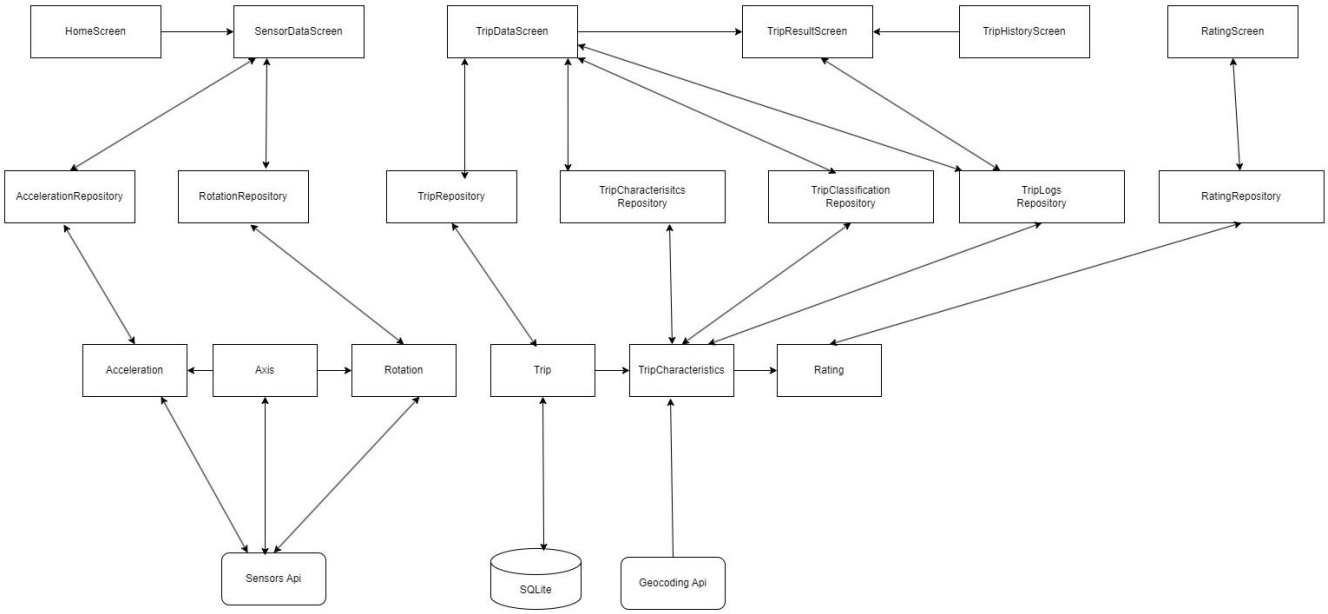


Рисунок 3.1 – Схема загальної структури системи

Як показано на схемі, система складається з джерел надходження і зберігання даних: СУБД SQLite, Geocoding Api та Sensors Api. Далі отримані дані обмінюються з рівнем моделей, які здійснюють конвертацію до необхідних типів для використання у програмі. Система містить наступні моделі: Axis, Acceleration, Rotation, Trip, TripCharacteristics та Rating. Джерела даних і моделі складають рівень даних у системі.

Наступний архітектурний рівень представляє доменний рівень, який складається з репозиторів, що приймають на вхід вже оброблені в моделях дані і

містять ряд методів, що дозволяють взаємодіяти з вище зазначеним сховищем даних та API. У системі використовуються AccelerationRepository, RotationRepository, TripRepository, TripCharacteristicRepository, TripLogsRepository, TripClassificationRepository та RatingRepository репозиторії.

Далі на схемі показана взаємодія доменного рівня з рівнем презентації у системі. Він складається з бізнес логіки, що реалізовано за допомогою використання підходу в управлінні станом BLoC, та екранів. Розроблена програма містить наступні екрани: HomeScreen, TripDataScreen, SensorDataScreen, TripResultScreen, RatingScreen та TripHistoryScreen. В свою чергу кожен з екранів, взаємодіє з компонентом, що відповідає за управління станом.

3.2 Набір необхідних даних для класифікації поведінки водія

Поведінка керування транспортним засобом представляє собою важливий аспект, описуючи, як водій маніпулює автомобілем у відповіді на дорожню обстановку та навколишнє середовище. Це ключовий фактор у проектуванні, розробці та впровадженні передових систем допомоги водієві [1] та інтелектуальних транспортних систем [2], які можуть бути впливовими через різноманіття факторів. Аналіз та вимірювання стилів водіння, які використовуються для вивчення можливих втручань, може істотно поліпшити комфорт водія та його загальний досвід.

У наш час практично у кожного є мобільний телефон, який слугує не лише засобом зв'язку, але й надзвичайно доступним інструментом. Отже, враховуючи цей факт, можна стверджувати, що смартфон є ідеальним знаряддям для вирішення нашої задачі. Завдяки вбудованим датчикам, починаючи від класичного акселерометра і закінчуючи недавно представленим датчиком солі, сучасний смартфон може надати широкий спектр функціональності.

Для реалізації потреб було обрано датчики, що відстежують рух автомобіля – акселерометр і гіроскоп. Було обрано ці два датчики для реалізації комплексного підходу, який найкраще відповідає реальним умовам, а також може використовуватися на широкому спектрі пристроїв.

На рисунку 3.2 наведено приклад автомобіля разом із трьома осями, які визначають рух: поздовжні, бічні та вертикальні. Сенсори – акселерометр та гіроскоп дозволяють точно відстежувати рухи вздовж осей X, Y і Z[8].

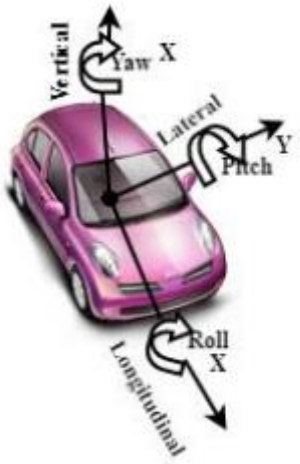


Рисунок 3.2 – Відстеження руху автомобіля

Активне керування автомобілем, що включає перевищення допустимої швидкості, раптові зупинки та стрімкі повороти уліво чи управо, може бути виявлене за допомогою реєстрації подій на датчиках акселерометра та гіроскопа. У світлі того, що сучасні користувачі практично всі мають смартфони із різноманітністю вбудованих датчиків, наша команда розробила програмне забезпечення для Android, яке використовує дані акселерометра та гіроскопа для виявлення агресивного стилю водіння.

На рисунку 3.3 зображений мобільний застосунок водія архітектури, що детально пояснює поведінку водія та роботу системи. Ця схема служить не лише оглядом для системи класифікації, але і включає в себе весь контекст застосування.

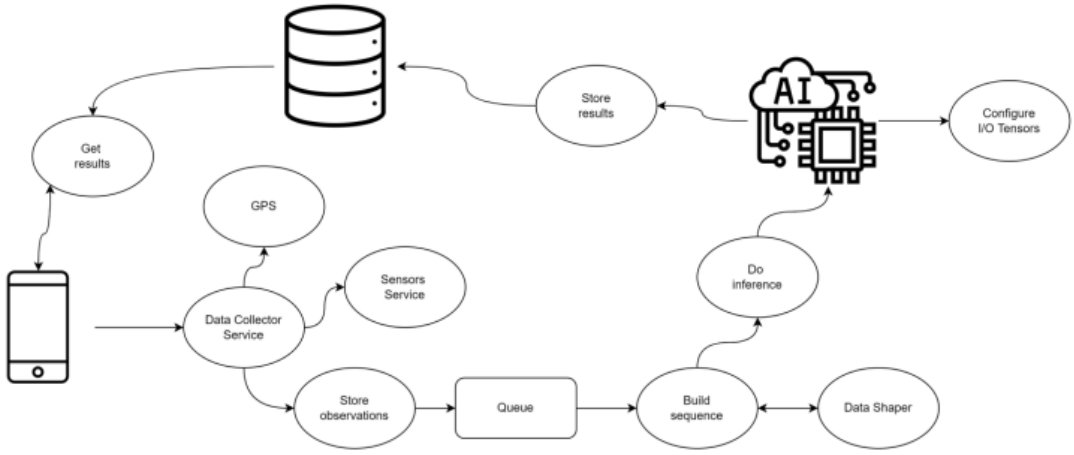


Рисунок 3.3 – Схема системи класифікації поведінки водія

Мобільний застосунок на смартфоні функціонує як служба збору даних, використовуючи різні датчики для збирання і збереження спостережень. За допомогою цих зібраних спостережень можна створити послідовність для подальшої класифікації[9].

Набір даних, що використовується в застосунку, включає в себе 8 різних функцій:

- 3 для вимірювання прискорення по кожній з осей (X, Y і Z в метрах за секунду в квадраті);
- 3 для вимірювання обертання (вісь X, Y, Z у градусах за секунду);
- мітку класифікації (повільна, нормальна або агресивна поведінка);
- мітку часу.

Зібрані дані оформлені у вибірках з частотою 2 зразка в секунду. Сила тяжіння була видалена з прискорення, і для збору інформації використовувались два основні датчики: акселерометр і гіроскоп.

Збір даних для тренування моделей машинного або глибокого навчання представляє собою важливий етап, оскільки результати напряму залежать від якості зібраних даних. Якість даних визначається тим, наскільки вони відповідають визначеній меті.

У цьому конкретному випадку було визначено, що поведінка водіння може бути класифікована як один із наступних типів:

- агресивна: раптові повороти ліворуч або праворуч. Прискорення та гальмування з великою силою;
- нормальна: середні події водіння, що відображають стандартну поведінку водія;
- повільна: підтримання швидкості нижче середньої.

Згідно з цими припущеннями, було проведено три поїздки: одну, де водіння було агресивним, іншу – нормальну, та третю – повільну, пройшовши ту ж саму ділянку дороги. Це дозволяє зібрати різноманітні дані, які відображають різні стилі водіння, і використовувати їх для подальшого навчання моделі.

3.3 Алгоритми застосовані для класифікації поведінки водія

Для прогнозування стилю водіння використовувались було проведено аналіз трьох різних алгоритмів: SoftmaxRegressor, DecisionTree і Knn від бібліотеки ml_algo для того щоб застосувати оптимальний алгоритм у програмі.

Таблиця 3.1 – Точність класифікації для різних алгоритмів алгоритмів

Алгоритм	Кількість класів	Точність
SoftmaxRegressor	2	0.724
	3	0.590
DecisionTree	2	0.795
	3	0.637
Knn	2	0.597
	3	0.491

У таблиці 3.1 наведені результати класифікації окремого екземпляра за даними датчика. Важливо відзначити, що такий підхід може бути не найкращим, оскільки стиль водіння є еволюційним і залежить від того, як довго водій робить різкі рухи транспортним засобом. Однак середнє значення за класифікацією може забезпечити хороший узагальнений огляд стилю водіння. Наприклад, можна розглядати маршрут тривалістю 10 хвилин, зі збором двох вибірок на секунду; кожен зразок класифікується як нормальний/агресивний або повільний/нормальний/агресивний. На кінці поїздки можна представити відсотковий розподіл стилю водіння, наприклад 40% нормальний і 60% агресивний або 20% повільний, 30% нормальний і 50% агресивний[10-12].

В результаті застосування алгоритмів було визначено точність їх роботи. На основі отриманих даних, було визначено, що для вирішення проблеми класифікації поведінки водія на дорозі підходить алгоритм «дерево рішень», так як точність його результатів була найвищою, тому він був застосований у програмі.

3.4 База даних системи

У зображеній ER-діаграмі бази даних системи на рисунку 3.4 видно важливий компонент, який виступає основою для забезпечення ефективної організації та надійного зберігання великої кількості інформації. Ця діаграма є ключовим елементом інфраструктури системи, яка об'єднує різні сутності та встановлює взаємозв'язки між ними.

Звернемо увагу на деталі структури бази даних, а саме ключові таблиці та їх опис. Цей аналіз дозволить краще зрозуміти, як дані в системі взаємодіють та як вони зберігаються для подальшого використання:

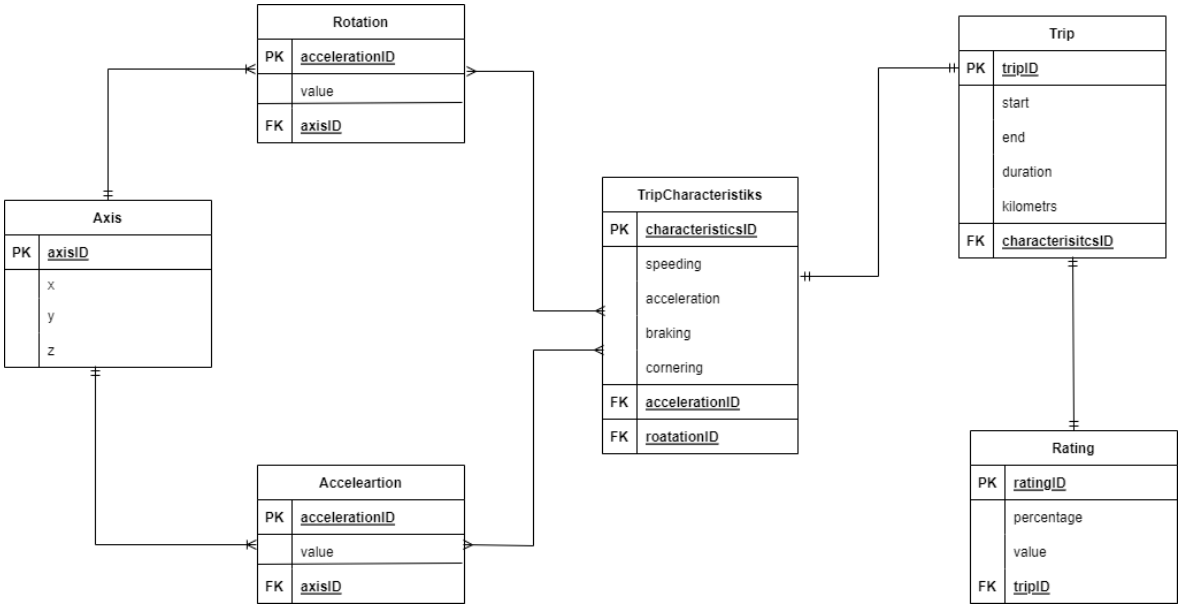


Рисунок 3.4 – ER діаграма бази даних системи

- axis – таблиця осей. В ній зберігаються значення даних, отриманих із сенсорів acceleromter та gyroscope по осях X, та Y та Z;
- acceleration – таблиця прискорення. Вона має зав’язок «багато-до-одного» з таблицею осей, що дає можливість отримати інформацію про прискорення на різних напрямках з accelerometer-сенсору;
- rotation – таблиця обертання. Вона має зав’язок «багато-до-одного» з таблицею осей, що дає можливість отримати інформацію про прискорення на різних напрямках з gyroscope-сенсору;
- trip – таблиця поїздок. Містить загальну інформацію по кожній поїздці, таку як початок поїздки, кінець, тривалість і кількість пройдених кілометрів. Має зв’язок «один-до-одного» відносно таблиці характеристики поїздки, що надає доступ до розширеної інформації про поїздку;
- trip_characteristics – таблиця характеристики поїздок. Містить інформацію про швидкість під час їзди, прискорення, гальмування та їзду на поворотах. Має зв’язки «багато-до-одного» з таблицями прискорення та обертання,

що дає доступ до даних із сенсорів, на основі яких в таблицю записуються дані про характеристику поїздки;

- rating – таблиця оцінки поїздки, яка зберігає дані про отримані бали про поїздку. Має зв’язок «один-до-одного» з таблицею поїздок.

3.5 Діаграма екранів User Flow

User flow – це концепція, яка визначає послідовність кроків, які користувачі виконують при взаємодії з продуктом чи сервісом. Діаграма екранів user flow є важливим інструментом в дизайні і розробці, оскільки вона відображає структуру та логіку користувацького інтерфейсу[17]. Розглянемо основні елементи та важливість діаграми екранів user flow:

- структура продукту. Діаграма екранів user flow допомагає визначити структуру продукту чи веб-сайту, визначаючи всі доступні екрани та їхні взаємозв'язки.
- послідовність взаємодій. Вона відображає послідовність кроків, які користувачі виконують для досягнення певної мети. Це може включати введення даних, навігацію, взаємодію з різними функціями тощо.
- взаємодії та функціонал. Кожен блок на діаграмі представляє окремий екран чи сторінку з конкретною функціональністю. Взаємодії користувача з елементами інтерфейсу та можливі дії відображаються у вигляді стрілок та коментарів.
- визначення ключових точок. Діаграма дозволяє виділити ключові точки, такі як етапи реєстрації, оплати чи взаємодії з важливим контентом. Це допомагає команді зосередитися на критичних аспектах досвіду користувача.

- тестування і оптимізація. Візуалізація user flow дозволяє команді дизайну та розробки легко ідентифікувати можливі нестачі в інтерфейсі та покращити їх, щоб забезпечити оптимальний користувацький досвід.
- комунікація в команді. Діаграма user flow є ефективним інструментом для комунікації в команді, допомагаючи розробникам, дизайнерам та іншим учасникам проєкту зрозуміти загальну логіку продукту.

Загалом, діаграма екранів user flow, що зображена на рисунку 3.5 визначає шлях користувача через продукт та допомагає забезпечити послідовність та зрозумілість взаємодій, сприяючи створенню ефективного та задовільного користувацького досвіду.

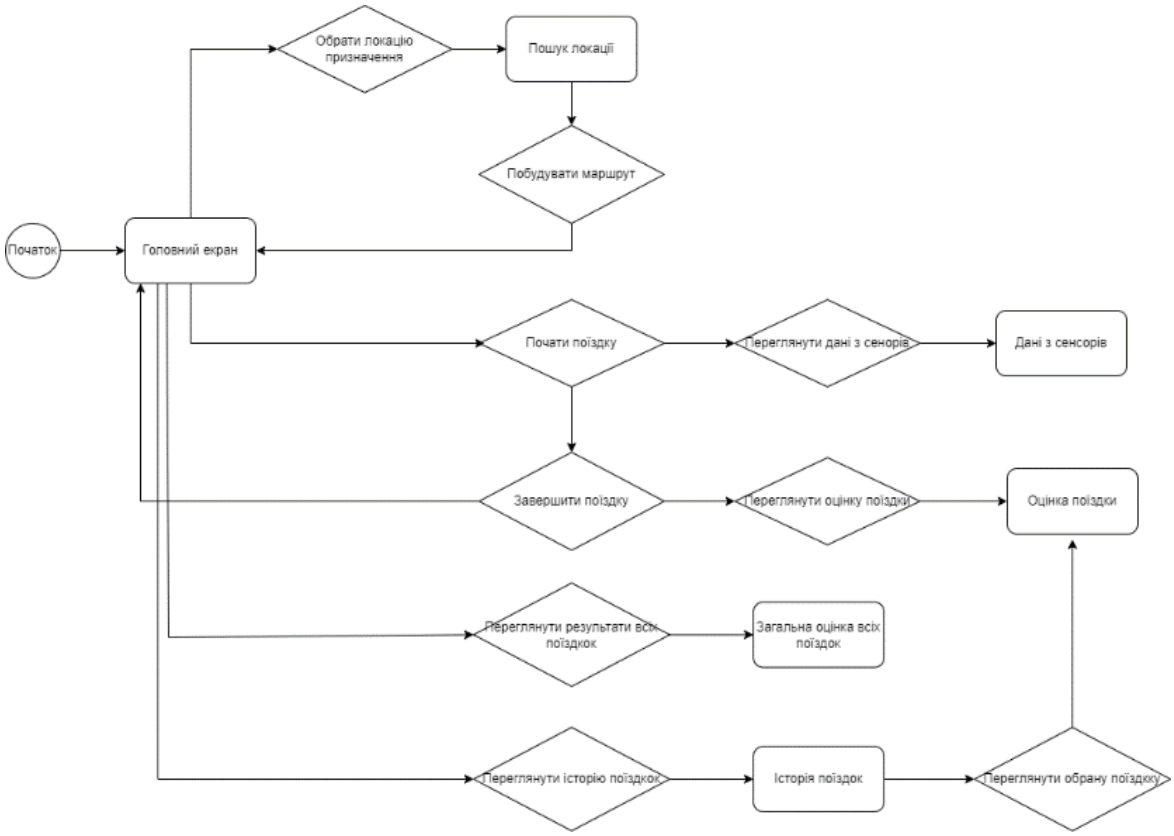


Рисунок 3.5 – Діаграма екранів User Flow

Нижче описані основні аспекти взаємодії користувача з програмою, представлені у діаграмі:

- головний екран – є ключовим екран у роботі програми і перший екран, куди потравляє користувач після запуску програми, тобто є стартовою точкою програми. Містить можливості переходу до функціональності пошуку адреси призначення, після чого будується маршрут та користувач знову повертається на головний екран;
- почати поїздку – дія користувача, що веде до можливості переглядання динамічних даних із сенсорів на відповідному екрані;
- завершити поїздку – дія користувача, яка робить доступною функцію «переглянути оцінку поїздки», що веде до відповідного екрану;
- переглянути результати всіх поїздок – дія, що веде до інтерфейсу загальної оцінки всіх поїздок;
- переглянути історію поїздок – дія користувача яка веде до екрану з історією поїздок, яка в свою чергу дає можливість переходу до інтерфейсу з детальною оцінкою кожної поїздки.

Висновки до розділу 3

У даному розділі було розглянуто реалізацію системи: її загальну структуру, моделі бази даних, діаграму екранів User Flow, необхідні дані для здійснення класифікації поведінки водія, застосований алгоритм класифікації. Було детально описано компоненти системи, які використовуються на різних архітектурних рівнях та представлено таблиці бази даних. Крім того, було описано типи класифікації поведінки водія. Також було представлено результати застосування трьох алгоритмів класифікації, а саме – визначення точності при вирішенні завдань класифікації. Алгоритм «дерево рішень» показав найкращі результати, в наслідок чого був використаний у реалізації системи.

4 ВЗАЄМОДІЯ КОРИСТУВАЧА З ПРОГРАМНОЮ СИСТЕМОЮ

4.1 Встановлення системи

Розроблена система підтримується на платформах Android та IOS, що робить її більш універсальною. Для встановлення застосунку користувачу необхідно мати apk-файл з програмою. Його можна згенерувати безпосередньо в Android Studio.

Існують два основних способи створення apk для Flutter-проєкту:

1) параметри збірки у інтегрованому середовищі розробки (IDE). У верхній частині панелі інструментів Android Studio перейдіть до Build < Flutter < Build APK як показано на рисунку 4.1. Далі Android Studio створить apk-файл для Flutter-проєкту;

2) команда терміналу: Введіть наступну команду в терміналі вашого редактора коду: flutter build apk --release.

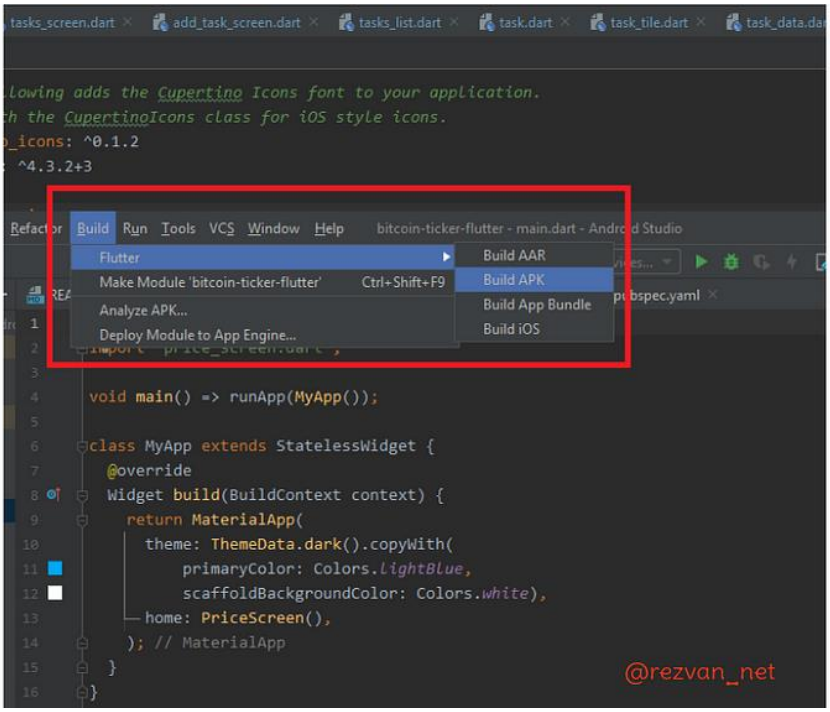


Рисунок 4.1 – Генерація APK через інтерфейс Android Studio

Завершивши цей процес, необхідно перейти до папки "build" у каталозі проєкту (зазвичай на диску C у Windows), знайдіть та перенести файл "app-release.apk" на свій мобільний пристрій, а потім встановити його, директорія якого показана на рисунку 4.2.

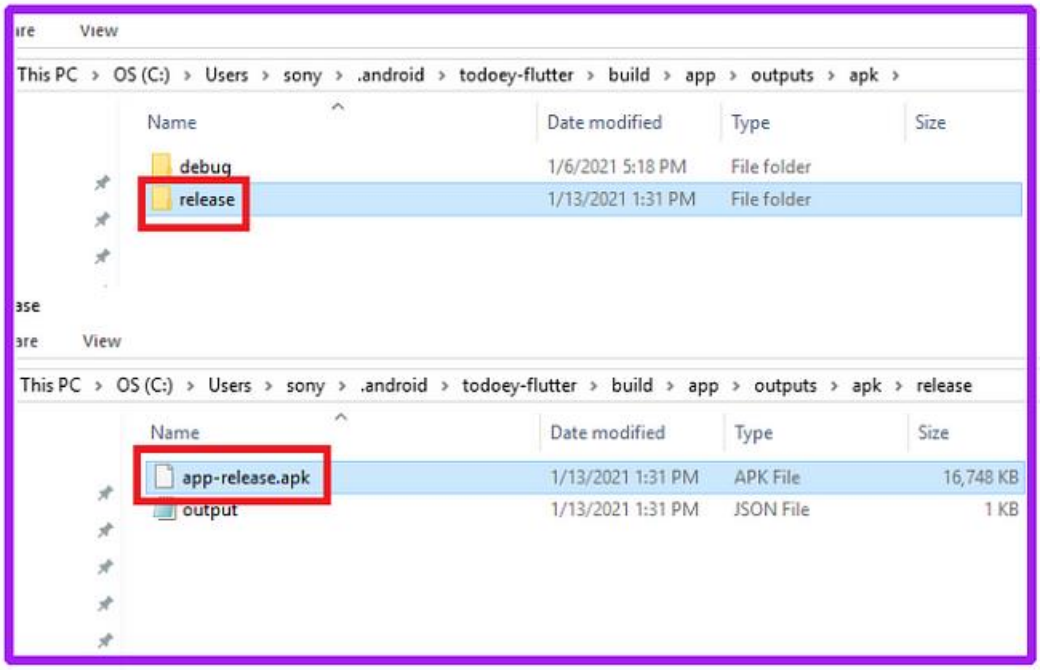


Рисунок 4.2 – Директорія виводу APK файлів

Після встановлення застосунку на мобільний телефон, користувачу необхідно надати дозвіл на використання геолокації, інакше застосунок не зможе відстежувати переміщення користувача, будувати маршрути та логувати необхідну інформацію.

4.2 Огляд можливостей системи

Після запуску програми, користувач потрапляє на головний екран, що представлений на рисунку 4.3, на якому зображена мапа. Вона містить основний функціонал, представлений у Google Maps Api, зокрема, відстеження геолокації користувача. Перед початком поїздки користувачу пропонується обрати локацію

призначення. Для цього необхідно натиснути кнопку «Search Places» у верхньому лівому кутку екрану та перейти в пошук локацій. При початку вводу назви локації пошук автоматично пропонує перелік можливих адрес. Після того як користувач обрав необхідну адресу, будується маршрут. Далі залишається почати рух. Для цього необхідно включити трекер відстеження поїздки. Він відображає переміщення по маршруту з можливістю перебудовувати його, якщо користувач змінює напрямок і надає можливість відстежувати дані з сенсорів телефону. Отже, при початку руху необхідно натиснути кнопку «Start Tracking».

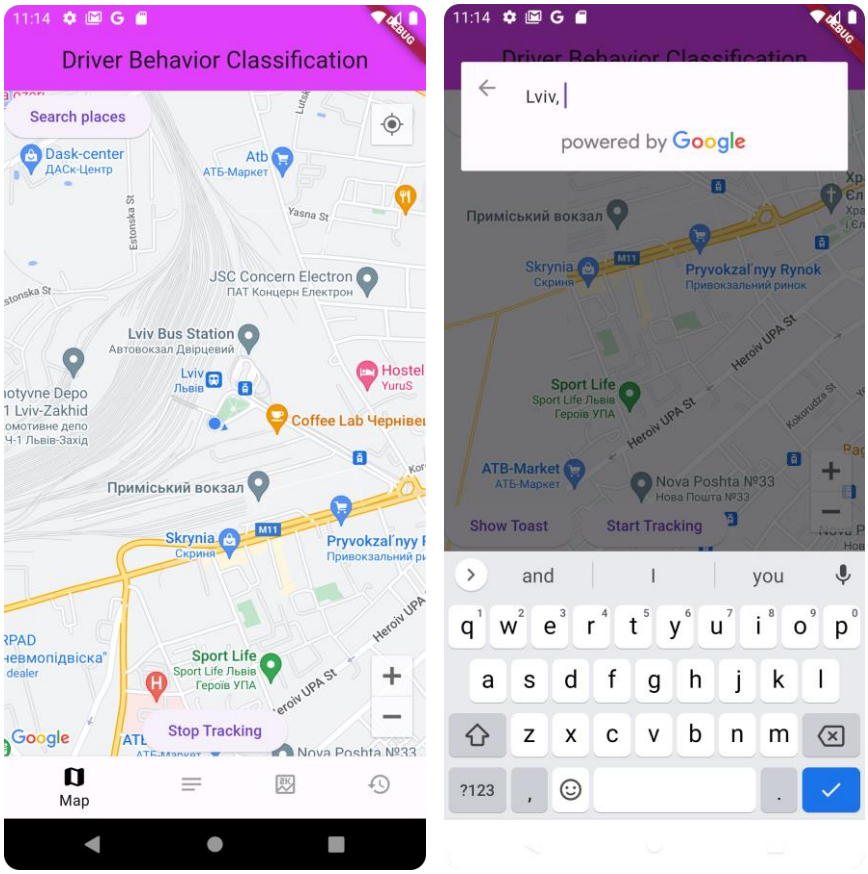


Рисунок 4.3 – Інтерфейс головного екрану застосунку

Після того, як рух розпочато стає доступною інформація на вкладці «Trip Data», що являє собою динамічні дані під час поїздки в реальному режимі, що зчитуються за допомогою сенсорів мобільного телефону (рис. 4.4).

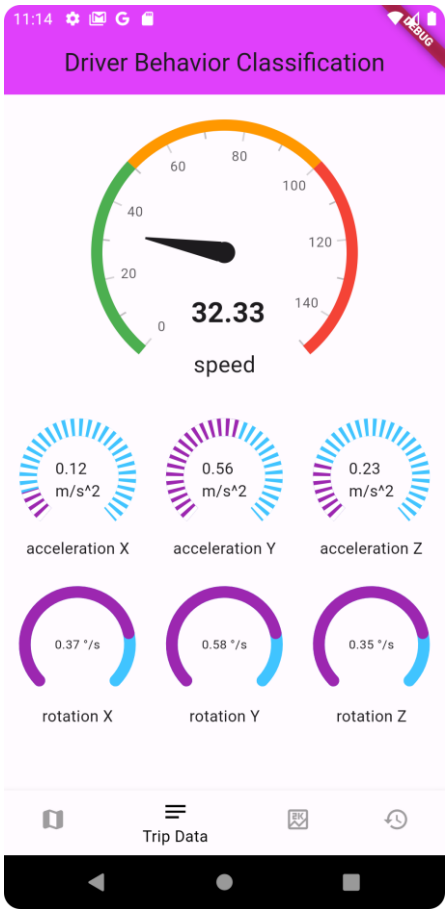


Рисунок 4.4 – Інтерфейс вкладки «Trip Data»

Як бачимо, під час поїздки збираються дані про швидкість транспортного засобу, прискорення та обертання по осі X, Y та Z, які є необхідними для класифікації поведінки водія на дорозі.

На основі вище описаних даних із сенсорів під час поїздки, програма робить оцінку вхідних даних і виводить користувачу різні зауваження або попередження при порушенні показників безпечного водіння, наприклад попередження про перевищення швидкості, різке гальмування, різка зміна прискорення тощо.

Як видно на рисунку 4.5 такі зауваження відображаються у вигляді діалогів, що є доступними на будь-якому екрані застосунку.

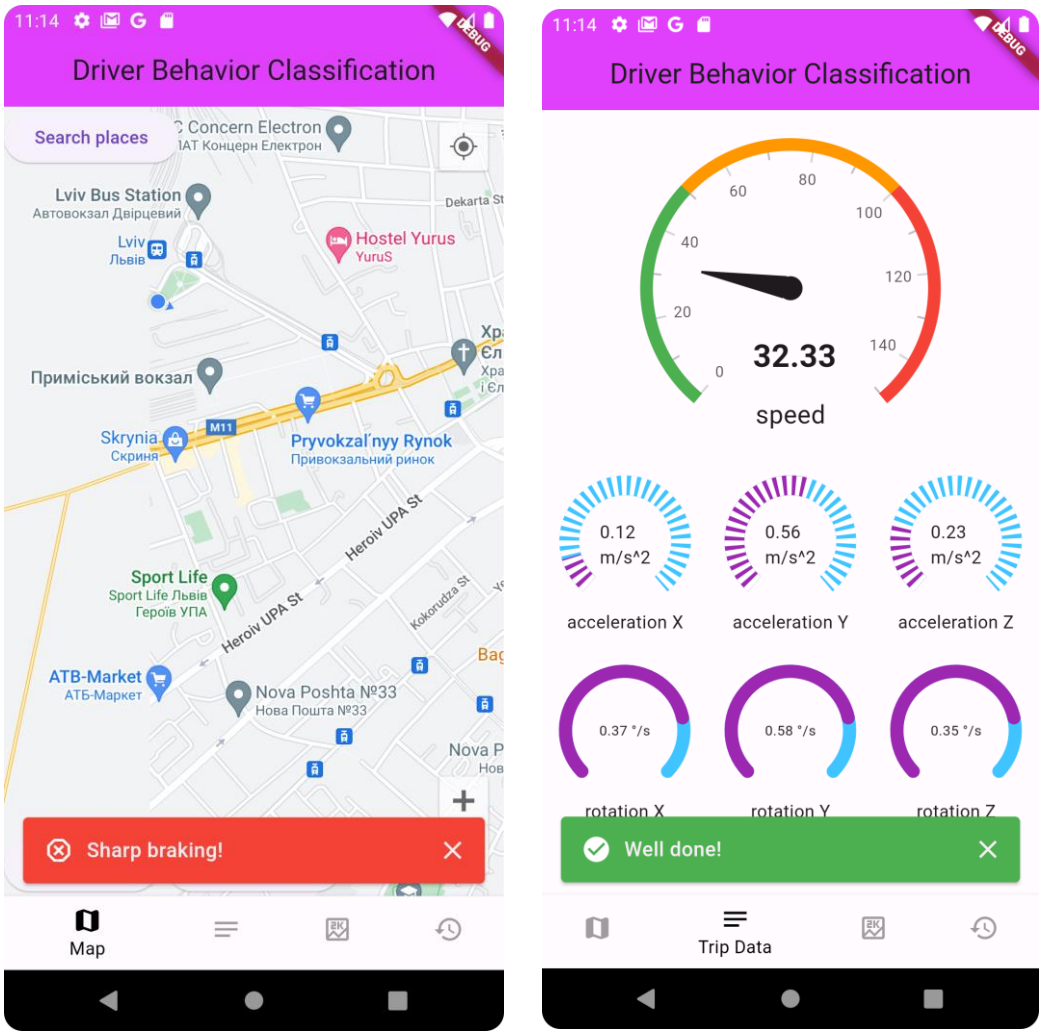


Рисунок 4.5 – Відображення зауважень або попереджень

Цей набір даних, який програма збирає під час поїздки, виявляється дуже важливим для класифікації поведінки водія на дорозі. Аналіз швидкості транспортного засобу, прискорення та обертання по трьох осях (X, Y, Z) дозволяє системі виявляти різноманітні аспекти водійської активності та вчасно реагувати на можливі порушення безпеки.

Якщо користувач хоче завершити поїздку, йому пропонується функція «Finish Trip» (рис. 4.6), яка зображена у вигляді кнопки на головному екрані знизу. Після завершення поїздки користувачу виводить відповідне діалогове вікно.

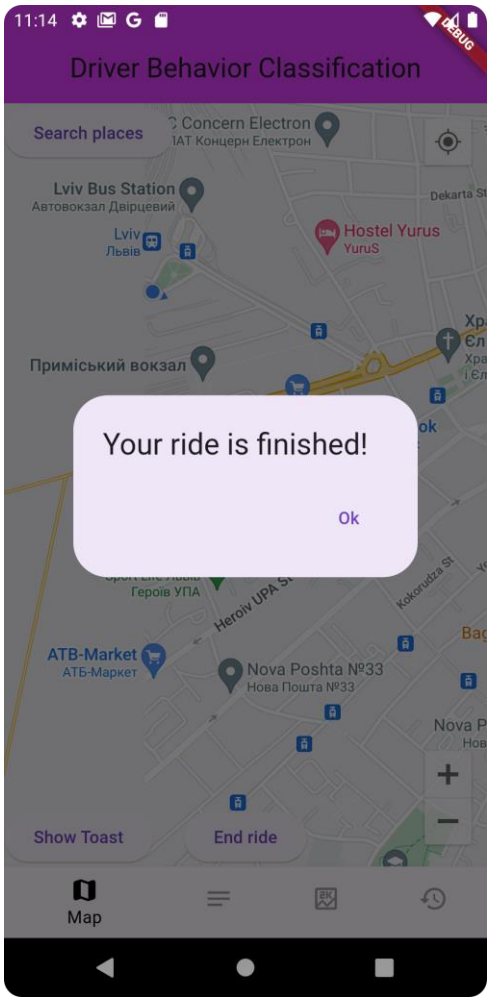


Рисунок 4.6 – Інтерфейс завершення поїздки

Після завершення поїздки користувач може переглянути її оцінку, де відображається результат обробленої інформації, зібраної із сенсорів мобільного телефону, як показано на рисунку 4.7. Спершу формуються загальні бали за поїздку, які рахуються на основі наступних критеріїв: швидкість під час поїздки, прискорення, гальмування, їзда на поворотах, відволікання на телефон. Далі вказується класифікація поведінки водія. Вона може бути нормальною, повільною або агресивною, що оцінюється відносно критеріїв безпечного водіння.

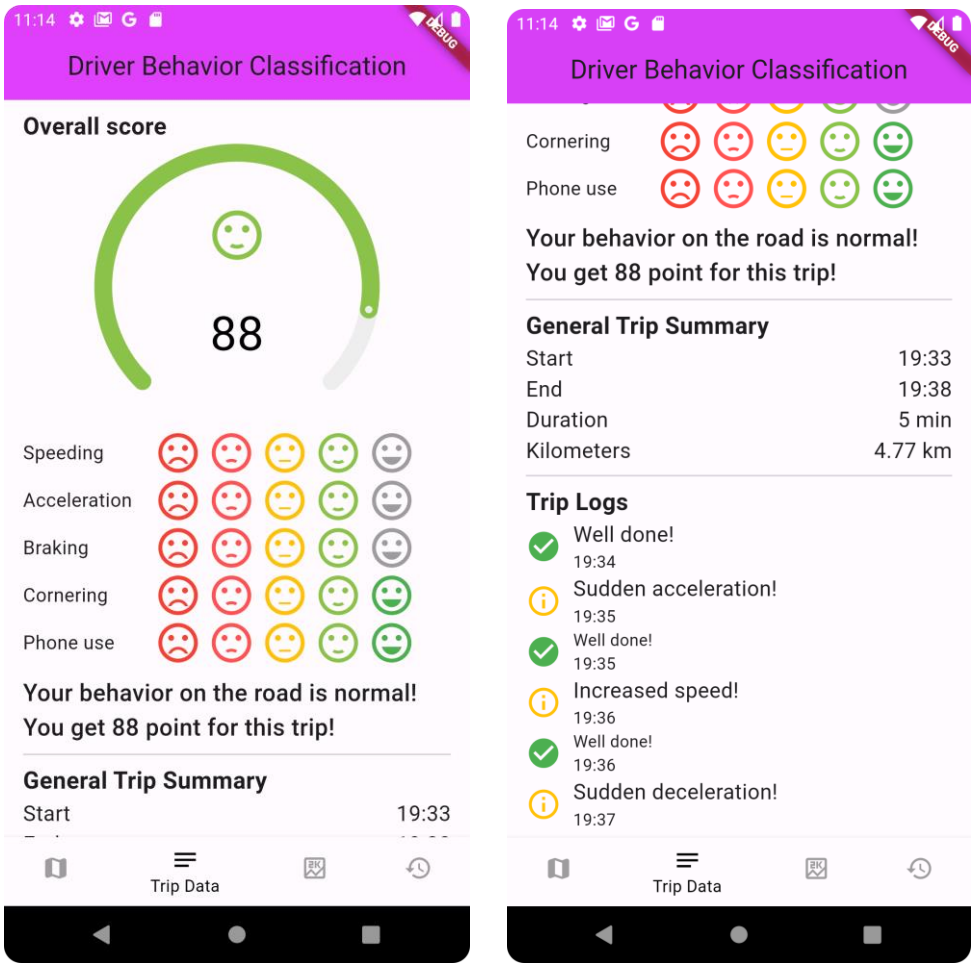


Рисунок 4.7 – Інтерфейс з оцінкою поїздки

Далі зібрана загальна інформація про поїздку: початок руху, кінець, тривалість і пройдені кілометри. Нижче на екрані зображені логи, зафіксовані під час поїздки, які відображаються по всьому застосунку, коли користувач ще не завершив рух. Таким чином водій може проаналізувати свої помилки під час водіння, що сприятиме вдосконаленню його навиків.

На вкладці «Score» (рис. 4.8) зображено загальну кількість балів, які користувач отримує під час поїздок. Також зображена інформація про кількість поїздок, пройдених кілометрів та проведених годин у дорозі. Крім того, користувачу пропонується ряд викликів, які сприяють набору загальних балів і свідчать про покращення навиків водіння, наприклад, проїхати 100 кілометрів, не перевищуючи швидкість. Такий підхід може мотивувати водія їздити обережніше.

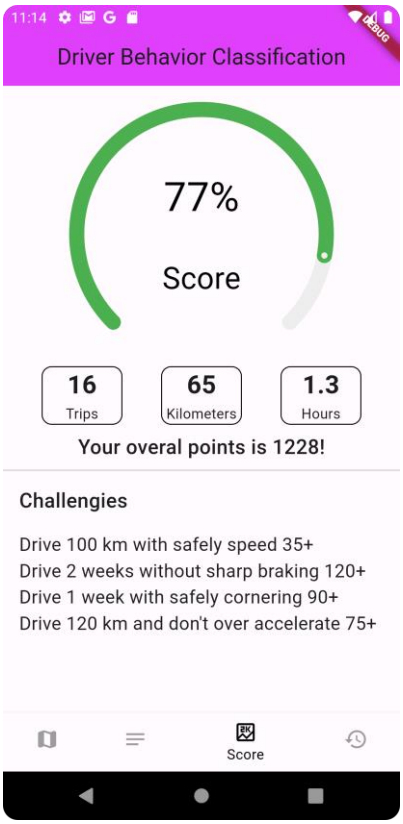


Рисунок 4.8 – Інтерфейс для загальної оцінки всіх поїздок

Також у програмі є вкладка «Trip History» (рис. 4.9), де користувачу пропонується список здійснених поїздок. Він може вибрати будь-яку поїздку із списку і переглянути детальну інформацію по кожній поїздці.

Така інтерактивна можливість перегляду історії поїздок не лише вдосконалює користувацький досвід, але й слугує практичним інструментом для аналізу власного використання транспорту. Користувач може використовувати цю інформацію для вдосконалення своєї поведінки на дорозі, оптимізації маршрутів та ефективного використання транспортних засобів. Наприклад, можливість детального вивчення поїздок може допомогти користувачеві визначити оптимальний час для подорожей, обрати найефективніший маршрут та визначити фактори, які впливають на витрати часу та ресурсів. В цілому, вкладка «Trip History» розширює функціонал програми, забезпечуючи користувачам зручний інструмент для аналізу та вдосконалення їхнього транспортного досвіду.

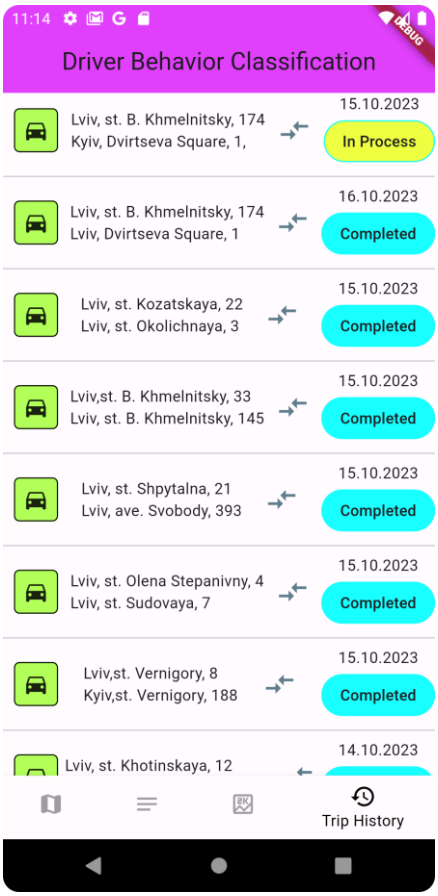


Рисунок 4.9 – Інтерфейс з історією всіх поїздок

Розглянувши уважно функціональність розробленої системи, визначено, що вона демонструє високий рівень ефективності та надійності у виконанні своїх завдань. Застосування передових технологій дозволило досягти відмінних результатів у взаємодії з користувачем.

Детальна інструкція користувачу включає в себе послідовні кроки та рекомендації щодо оптимального використання системи. Користувачам надається доступ до всіх необхідних ресурсів та функціоналу, спрощуючи їхні завдання та підвищуючи загальний комфорт використання.

Призначена для широкого кола аудиторії, система акцентує на простоті та інтуїтивності інтерфейсу, забезпечуючи зручний доступ до усіх можливостей. Це створює сприятливі умови для користувачів різного рівня досвіду, дозволяючи їм швидко оволодіти всіма перевагами системи.

Висновки до розділу 4

В даному розділі було розглянуто функціональність системи та надано інструкцію користувачу по її встановленню. Розроблене програмне забезпечення має наступні можливості: робота з картою, що включає пошук адрес призначення, побудову маршрутів, відстеження переміщення по маршрутах, запуск трекера даних із сенсорів під час поїздки, вивід зауважень та попереджень для водія під час поїздки, представлення оцінки і логів про поїздку, перегляд всіх поїздок з їх оцінками.

Детальна інструкція користувачу включає в себе послідовні кроки та рекомендації щодо оптимального використання системи. Користувачам надається доступ до всіх необхідних ресурсів та функціоналу, спрощуючи їхні завдання та підвищуючи загальний комфорт використання.

Призначена для широкого кола аудиторії, система акцентує на простоті та інтуїтивності інтерфейсу, забезпечуючи зручний доступ до усіх можливостей. Це створює сприятливі умови для користувачів різного рівня досвіду, дозволяючи їм швидко оволодіти всіма перевагами системи.

5 РОЗРОБКА СТАРТАПУ ПРОЄКТА

Даний розділ присвячений вивченню ключових аспектів і можливостей створення інноваційного стартапу, спрямованого на управління та оптимізацію процесів розробки проєктів. У світлі стрімкого технологічного розвитку та зростання обсягів проєктної діяльності, студентських та наукових колективів університетів стає важливим завданням забезпечення ефективного та організованого управління цими процесами.

У даному розділі ми розглянемо актуальні аспекти створення та розвитку стартапу, спрямованого на розв'язання завдань, пов'язаних із сучасними університетськими проєктами. Аналіз ринкових тенденцій, вивчення потреб користувачів у сфері університетської розробки та визначення ключових вимог дозволять нам сформулювати стратегію стартапу та визначити напрямки подальшого дослідження.

В ході розділу буде приділено увагу не лише технічним аспектам створення продукту, але й взаємодії з університетським середовищем, включаючи можливості партнерства та інтеграції інновацій у процеси освіти та досліджень. Надалі розглядатимуться стратегії впровадження та популяризації стартапу, а також практичні аспекти його функціонування в умовах високоосвітнього сегменту.

5.1 Опис ідеї проєкту

Розглянемо концепцію представленого проєкту, вивчимо можливі напрямки його використання та переваги для користувачів, узагальнюючи цю інформацію у таблиці 5.1

Таблиця 5.1 – Опис ідеї проєкту

Зміст ідеї	Напрямки застосування	Вигоди для користувачів
1	2	3
Розробка програмного продукту, який використовує сучасні технології для класифікації поведінки водія на дорозі	Безпека на дорозі	Забезпечення особистої безпеки та зниження ризику аварій шляхом виявлення агресивного водіння та отримання рекомендацій щодо поліпшення стилю водіння
	Страхування та транспортні компанії	Можливість отримання індивідуальних та адаптованих страхових тарифів на підставі реальних даних стилю водіння
	Освіта водіїв	Отримання інформації про власний стиль водіння та персоналізовані поради для покращення безпеки та зручності

Продовження таблиці 5.1

1	2	3
	Дослідження та статистика	Дослідження та статистика Своєчасне інформування про потребу в технічному обслуговуванні автомобіля для забезпечення надійності та тривалого терміну служби
	Технічне обслуговування	

З метою техніко-економічного аналізу проєкту важливо виконати порівняльний огляд з конкурентами, ідентифікувати сильні та слабкі сторони розробленої системи та вжити заходів для максимізації переваг та мінімізації недоліків[18]. У таблиці 5.2 надано детальний огляд порівняльних характеристик проєкту в порівнянні з аналогічними рішеннями на ринку. Цей аналіз слугує фундаментом для розуміння конкурентних переваг та позиціонування нашого проєкту в екосистемі ринку.

В рамках техніко-економічного аналізу проєкту вирішальну роль відіграє порівняльний огляд з існуючими конкурентами, оскільки це надає можливість ідентифікувати сильні та слабкі сторони нашої розробленої системи. Аналіз конкурентної обстановки дозволяє ефективно визначити найбільш перспективні напрямки для вдосконалення та вдосконалення проєкту. Цей аналіз є стратегічно важливим етапом, оскільки він становить фундамент для розуміння конкурентних переваг, а також визначення оптимальної позиції нашого проєкту в екосистемі ринкових відносин.

Таблиця 5.2 – Визначення сильних, слабких та нейтральних характеристик

№	Характеристика	Розроблена система	Конкуренти	Результат
1	2	3	4	5
1	Технічна можливість	Всі необхідні технології є доступними для розробки системи	Конкуренти вже використовують налагоджені системи, які можна інтегрувати	Сильна
2	Користувацький інтерфейс	Система містить зручний інтерактивний інтерфейс	Деякі з конкурентів мають зручний і гарний інтерфейс	Сильна
3	Множинна функціональність	Система вирішує ряд задач, що робить застосунок привабливим для користувачів	На ринку вже є додатки, що пропонують багато функціональних можливостей	Сильна
4	Конкуренція на ринку	Система враховує можливість аналогічних і пропонує додатковий функціонал	Подібні системи на ринку вже існують	Слабка

Продовження таблиці 5.2

1		2	3	4	5
6		Потенційна аудиторія	Націлена на водіїв транспортних засобів	Конкуренти враховують побажання аудиторії	Нейтральна
7		Технологічна база	Використовує доступні технології	Використовують доступні технології для своїх систем	Нейтральна

Ця таблиця демонструє, чи система, яку ми розробляємо, виходить вперед за ключовими характеристиками порівняно з конкурентами. Якщо значення в колонці "Сильна", то наша система переважає конкурентів за цією характеристикою. Якщо "Слабка" – переваги відсутні, а якщо "Нейтральна", то системи приблизно рівні за цією характеристикою. Аналізуючи таблицю, можна визначити, що у нашої системи є як сильні, так і слабкі сторони, проте переваг переважає, що надає сенс спробувати впровадження стартап-проєкту.

5.2 Технологічний аудит проєкту

Для оцінки реалізаційних перспектив стартап-проєкту необхідно ретельно вивчити наявні технології та їхню доступність. У таблиці 5.3 представлено огляд технологій, які можуть бути використані для реалізації проєкту. Проаналізувавши ці дані, ми можемо зробити висновки щодо того, наскільки ефективно та реалізовано буде впровадження проєкту, залежно від обраних технологій.

Таблиця 5.3 – Технологічна здійсненність стартап проєкту

№	Складова проєкту	Технології реалізації	Наявність технологій	Доступність технологій
1	Сховище даних	СУБД SQLite	Наявна	Доступна і безкоштовна
2	Засоби розробки	Мова програмування Dart, фреймворк Flutter, середовище розробки Android Studio	Наявні	Доступні і безкоштовні
3	Робота з мапами	Google Maps Api, плагін Geolocation, плагін Geocoding	Наявні	Доступні, але безкоштовні лише пробний період
4	Отримання даних, необхідних для класифікації поведінки водія	Бібліотека sensor_plus від фреймворку Flutter	Наявна	Доступна та безкоштовна
5	Застосування штучного інтелекту	Бібліотека ml_algo від фреймворку Flutter	Наявна	Доступна та безкоштовна
6	Публікація застосунку	Платформи Google Play та App Store	Наявні	Доступні та платні

Із представленою в таблиці 5.3 інформацією очевидно, що всі необхідні технології для успішної реалізації стартап-проєкту є наявними. Більшість з цих технологій є відкритими та доступними безкоштовно, але важливо врахувати, що деякі інструменти можуть вимагати плату за використання. Незважаючи на це, наявність альтернатив та стратегій для ефективного використання ресурсів дозволяє подолати цю обмеженість[19]. Таким чином, можемо зробити висновок, що реалізація стартап-проєкту технічно можлива та перспективна.

З огляду на дані в таблиці, можна зазначити, що всі технології, необхідні для втілення стартап-проєкту, доступні, більшість з них є відкритими та безкоштовними, але є окремі платні опції. Враховуючи це, можна зробити висновок, що впровадження стартап-проєкту є можливим.

5.3 Аналіз ринкових можливостей стартап-проєкту

Перед впровадженням стартап-проєкту необхідно ретельно проаналізувати ринкові можливості, які сприятимуть ефективному впровадженню проєкту, а також ретельно вивчити можливі загрози, які можуть вплинути на його успішність. У рамках цього аналізу слід приділити увагу стану ринку, виявленню потреб потенційних клієнтів, аналізу пропозицій та можливостей, які надають конкуренти. Ці важливі аспекти дозволять визначити стратегії впровадження та конкурентні переваги стартапу. Детальний огляд стану ринку та його характеристик наведено в таблиці 5.4.

Аналіз ринкових можливостей охоплює вивчення стану ринку, виявлення потреб та вимог потенційних клієнтів. Розгляд цих аспектів необхідний для розроблення ефективних стратегій впровадження та позиціонування стартапу в конкурентному середовищі.

Таблиця 5.4 – Характеристика ринку стартап проєкту

№	Показник стану ринку	Характеристика
1	Обсяг ринку	100+
2	Динаміка зростання	Стрімка
3	Маржинальність	38 %
4	Рівень конкуренції	Висока

Базуючись на попередній оцінці ринку та його характеристиках, виявлено суттєві виклики, що можуть ускладнити успішне впровадження стартапу[20]. Найбільш значущими є висока конкуренція на ринку та обмежена динаміка зростання через відсутність нових телекомунікаційних провайдерів. Крім того, хоча рентабельність перевищує банківські ставки, вона залишається на невеликому рівні. Наступним кроком буде ідентифікація потенційних клієнтів, їхніх характеристик та вивчення їхніх вимог для подальшого розроблення стратегії впровадження на ринку.

Таблиця 5.5 – Характеристика потенційних клієнтів

№	Потреби ринку	Цільова аудиторія	Вимоги клієнтів
1	2	3	4
1	Автоматизована класифікація поведінки водія на дорозі	Водії транспортних засобів	Інтегрована система, яка автоматично аналізує стиль водіння, забезпечуючи точні результати та деталізацію

Продовження таблиці 5.5

1	2	3	4
2	Високоточний моніторинг та аналіз даних з сенсорів мобільного телефону	Технологічно орієнтовані користувачі та фахівці з області транспортної безпеки	Можливість отримання детальної інформації з датчиків телефону для вивчення та вдосконалення власного стилю водіння
3	Взаємодія та інтеграція з іншими технологічними рішеннями в автомобільній сфері	Виробники автомобілів та розробники автотехнологій	Можливість взаємодії програмного продукту з іншими системами автомобіля, такими як системи безпеки, електроніка автомобіля та системи управління, забезпечуючи повноцінну інтеграцію в екосистему транспортних засобів

Були проведені аналіз потреб потенційних клієнтів з метою визначення того, наскільки задоволені вони реалізованою системою[21]. Цей огляд дозволив виокремити пріоритетні аспекти для подальших покращень та удосконалень. Важливо враховувати не лише те, що вже реалізовано в системі, а й на що необхідно зосередити увагу, щоб максимально відповідати очікуванням клієнтів.

Також були розглянуті фактори загроз та можливостей, які можуть вплинути на подальший розвиток проєкту[22]. Цей аналіз, представлений у таблицях 5.6 та

5.7, становить стратегічну основу для визначення шляху подальших дій та забезпечення успішного взаємодії з динамічними умовами ринку.

Таблиця 5.6 – Фактори загроз стартап проєкту

№	Фактор	Зміст загрози	Реакції клієнта
1	Конкуренція на ринку	Поява нових конкурентів або посилення позицій існуючих	Пошук інновацій та унікальних можливостей для відзначення серед конкурентів, можливе вдосконалення функціональності та розширення сервісів
2	Технічні проблеми	Збої в роботі системи, помилки та інші технічні проблеми	Забезпечення оперативного та ефективного технічного обслуговування, покращення стабільності системи
3			

Таблиця 5.7 – Фактори можливостей стартап проєкту

№	Фактор	Зміст можливості	Реакція клієнта
1	Розширення ринку	Захоплення нових сегментів ринку або введення продукту на нові території	Розвиток стратегій маркетингу та реклами для привертання нових клієнтів, розширення функціональних можливостей
2	Технологічні інновації	Запровадження новітніх технологій та покращення функціональності продукту	Інвестиції в дослідження та розвиток
3	Партнерські можливості	Укладання стратегічних партнерських угод та співпраця з іншими компаніями для розширення ринкових можливостей та забезпечення ефективного впровадження проєкту	Підтримка активної комунікації з партнерами, участь у переговорах та укладанні вигідних угод, надання підтримки для успішної реалізації партнерських ініціатив

Ключовим завершальним кроком у вивченні ринкових можливостей є розроблення SWOT-аналізу, який включає в себе виділення сильних (Strengths) та

слабких (Weaknesses) сторін проєкту, а також можливостей (Opportunities) та загроз (Threats). Результати цього аналізу відображені в таблиці 5.8.

Таблиця 5.8 – Матриця SWOT аналізу

Сильні стороні	Слабкі стороні
Використання сучасних технологій та відомих алгоритмів для класифікації поведінки водія. Велика кількість доступних та безкоштовних технологій для реалізації проєкту. Можливість взаємодії з користувачами через інтуїтивний інтерфейс мобільного застосунку.	Існування великої конкуренції в галузі подібних мобільних додатків. Обмежена рентабельність проєкту, порівняно з іншими видами інвестицій. Необхідність постійного оновлення даних та алгоритмів для підтримки актуальності системи.
Можливості	Загрози
Зростання інтересу до безпеки на дорозі та водійської поведінки. Розвиток та популяризація сучасних мобільних технологій. Розширення функціоналу застосунку для привертання нових користувачів.	Зміни в законодавстві, які можуть обмежити можливості збору та обробки даних водіїв. Несприятливі економічні умови, які можуть вплинути на інвестиційну привабливість проєкту. Зміна попиту на подібні мобільні додатки через зміну тенденцій та покликань ринку.

Висновки до розділу 5

У даному розділі було представлено комплексний погляд на ідею проєкту, включаючи технологічний аудит та глибокий аналіз ринкових можливостей. Висновки, які можна зробити з урахуванням наведених вище даних, свідчать про наявність значущих викликів та загроз на шляху впровадження стартапу. Основними серйозними перешкодами виявилися насичений ринок, відсутність нових учасників у галузі, висока конкурентоспроможність та обмежена рентабельність. Ці фактори ускладнюють перспективи успіху стартапу, навіть у звіті про можливості, які були виявлені.

Зазначені висновки свідчать про те, що впровадження даного стартапу буде великим викликом через складні умови ринку. Насиченість галузі та висока конкурентоспроможність можуть стати значущими бар'єрами для успіху проєкту. Однак, недивлячись на це, результати технологічного аудиту вказують на наявність інноваційних можливостей, які можуть стати ключовими конкурентними перевагами стартапу.

Додатково, важливо врахувати, що, незважаючи на виявлені труднощі, створення ефективної стратегії маркетингу та партнерських відносин може визначити успіх проєкту в умовах високої конкуренції. Важливим кроком може бути співпраця з ключовими гравцями галузі та активне впровадження інноваційних рішень для привертання уваги та відзначення на ринку. Такий підхід може допомогти подолати виявлені труднощі та забезпечити стартапу стійкість у конкурентному середовищі.

ВИСНОВКИ

У ході виконання магістерської роботи було розроблено мобільний застосунок для класифікації поведінки водія на дорозі з можливістю відображення зауважень та попереджень у разі відхилення показників із сенсорів від норми та здійснення оцінки поїздки.

Для розробки системи було проаналізовано аналогічні додатки, такі як Waze, DriveScore. Було виявлено їх переваги і недоліки, які було взято до уваги при проектуванні системи. Зокрема, багато додатків аналізують стан навколишнього середовища, але не аналізують безпосередньо поведінку водія на дорозі. Також інформація про поїздку не подається достатньо детально для користувача. Ці недоліки було усунено в розробленій системі.

В роботі було застосовано наступні технології: мова програмування Dart, фреймворк Flutter, середовище розробки Android Studio. Для роботи з картами було використано Google Maps Api, geocoding плагін та geolocation плагін. Для зчитування даних із сенсорів мобільного телефону під час поїздки було застосовано пакет sensors_plus, а для визначення поведінки водія на дорозі – бібліотеку ml_algo.

Також при проектуванні системи було проведено аналіз різних архітектур і обрано багатошарову архітектуру для розробки системи, що розділяє компоненти програми на наступні рівні: рівень даних, доменний рівень та рівень презентації. В якості підходу управління станом було обрано підхід BLoC.

В результаті розроблене програмне забезпечення вирішує наступні проблеми:

- визначення місцезнаходження користувача за системою GPS;
- побудова маршрутів для навігації водія;
- збір даних із сенсорів під час руху транспортного засобу;
- відображення зауважень та попереджень під час поїздки;
- вивід даних про поведінку водія на дорозі;
- вивід оцінки поїздки відносно отриманих даних про поведінку водія;
- вивід загальної оцінки всіх поїздок протягом користування додатком.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Dart’s core libraries. URL: https://dart.dev/libraries/ (дата звернення: 15.1.2023).
2. Flutter API reference documentation. URL: https://api.flutter.dev/ (дата звернення: 18.11.2023).
3. Flutter Bloc: An introduction. URL: https://medium.com/codex/flutter-bloc-an-introduction-with-cubit-7eae1e740fd0/ (дата звернення: 18.11.2023).
4. Sensors plus pub dev package. URL: https://pub.dev/packages/sensors_plus/ (дата звернення: 21.11.2023).
5. Geocoding pub dev package. URL: https://pub.dev/packages/geocoding/ (дата звернення: 25.11.2023).
6. Geolocator pub dev package. URL: https://pub.dev/packages/geolocator/ (дата звернення: 08.12.2023).
7. Wang R. AdaBoost for feature selection, classification and its relation with SVM, a review, 2012, 807 p.
8. Fabio M., Marulli F., Mercaldo F. Santone A: Neural networks for driver behavior analysis, 2012, 280 p.
9. Dong, S., Zhou, J. A Comparative Study on Drivers’ Stop/Go Behavior at Signalized Intersections Based on Decision Tree Classification Model. Journal of Advanced Transportation, 2020, 13 p.
10. Pathivada, B.K., Perumal, V. Modeling Driver Behavior in Dilemma Zone under Mixed Traffic Conditions. Transportation Research Procedia, 2017, 968 p.
11. Chawla, N.V., Bowyer, K.W., Hall, L.O., Kegelmeyer, W.P. SMOTE: Synthetic Minority Over-sampling Technique. Journal of Artificial Intelligence, 2018, 322 p.
12. Precht L., Keinath A., Krems J.F. Effects of driving anger on driver behavior—Results from naturalistic driving data. Transp. Res. Part F Traffic Psychol. Behav, 2017, 45 p.

13. Kadoya Y., Watanapongvanich S., Khan M.S.R. How is emotion associated with driving speed? A study on taxi drivers in Japan. Transp. Res. Part F Traffic Psychol. Behav, 2021, 79 p.
14. Bernstein J.P.K., DeVito A., Calamia M. Associations between emotional symptoms and self-reported aberrant driving behaviors in older adults. Accid. Anal. Prev, 2019, 127 p.
15. Singh G., Bansal D., Sofat S. A smartphone based technique to monitor driving behavior using DTW and crowdsensing. Pervasive Mob. Comput, 2017, 40 p.
16. Wu M., Zhang S., Dong Y. A Novel Model-Based Driving Behavior Recognition System Using Motion Sensors. Sensors, 2016, 16 p.
17. Xie J., Hilal A.R., Kulic D. Driving Maneuver Classification: A Comparison of Feature Extraction Methods. IEEE Sens. J, 2018, 18 p.
18. Shahverdy M., Fathy M., Berangi R., Sabokrou M. Driver behavior detection and classification using deep convolutional neural networks. Expert Syst. Appl, 2020, 149 p.
19. Castignani G., Derrmann T., Frank R., Engel T. Driver Behavior Profiling Using Smartphones: A Low-Cost Platform for Driver Monitoring. IEEE Intell. Transport. Syst. Mag, 2015, 91 p.
20. Brombacher P., Masino J., Frey M., Gauterin F. Driving event detection and driving style classification using artificial neural networks; Proceedings of the 2017 IEEE International Conference on Industrial Technology; Toronto, ON, Canada, 2017, 997 p.
21. Li G., Wang Y., Zhu F., Sui X., Wang N., Qu X., Green P. Drivers' visual scanning behavior at signalized and unsignalized intersections: A naturalistic driving study in China. J. Saf. Res, 2019, 71 p.
22. Ansar M.S., Ma Y., Chen S., Tang K., Zhang Z. Investigating the trip configured causal effect of distracted driving on aggressive driving behavior for e-hailing taxi drivers. J. Traffic Transp, 2021, 123 p.

ДОДАТКИ

ДОДАТОК А Лістінг розробленої програми

```
void main() {
  final GoogleMapsFlutterPlatform mapsImplementation =
      GoogleMapsFlutterPlatform.instance;
  Completer<AndroidMapRenderer?>? _initializedRendererCompleter;

  Future<AndroidMapRenderer?> initializeMapRenderer() async {
    if (_initializedRendererCompleter != null) {
      return _initializedRendererCompleter!.future;
    }

    final Completer<AndroidMapRenderer?> completer =
    Completer<AndroidMapRenderer?>();
    _initializedRendererCompleter = completer;

    WidgetsFlutterBinding.ensureInitialized();

    final GoogleMapsFlutterPlatform mapsImplementation =
        GoogleMapsFlutterPlatform.instance;

    if (mapsImplementation is GoogleMapsFlutterAndroid) {
      unawaited(mapsImplementation
          .initializeWithRenderer(AndroidMapRenderer.latest)
          .then((AndroidMapRenderer initializedRenderer) =>
          completer.complete(initializedRenderer)));
    } else {
      completer.complete(null);
    }

    return completer.future;
  }

  if (mapsImplementation is GoogleMapsFlutterAndroid) {
    mapsImplementation.useAndroidViewSurface = true;
    initializeMapRenderer();
  }
  runApp(const MyApp());
}


class MyApp extends StatefulWidget {
  const MyApp({super.key});

  @override
  State<MyApp> createState() => _MyAppState();
}

class _MyAppState extends State<MyApp> {

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
```

```
        colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),
        useMaterial3: true,
      ),
      home: HomeScreen(),
    );
  }
}

class PolylineModel {
  List<LatLng> polylineCoordinates;
  Set<Polyline> polyline;
  PolylineModel({
    required this.polylineCoordinates,
    required this.polyline,
  });
}

class HomeScreen extends StatefulWidget {
  HomeScreen({super.key});

  @override
  State<HomeScreen> createState() => _HomeScreenState();
}

class _HomeScreenState extends State<HomeScreen> {
  late Completer<GoogleMapController> googleMapsController = Completer();
  final homeScaffoldKey = GlobalKey<ScaffoldState>();
  final searchScaffoldKey = GlobalKey<ScaffoldState>();

  Timer? snackBarTimer;

  @override
  void initState() {
    // TODO: implement initState
    super.initState();
    WidgetsBinding.instance.addPostFrameCallback((_) async {
      DirectionsService.init(apiKey);
    });
    _getCurrentPosition();
    getReady();
  }

  Position? currentPosition;
  Location? destinationPoint;

  Set<Marker> markers = {};

  Future<void> _getCurrentPosition() async {
    LocationPermission permission;
    permission = await Geolocator.requestPermission();
    currentPosition = await Geolocator.getCurrentPosition(desiredAccuracy:
LocationAccuracy.high);
  }

  final destCtrl = TextEditingController();

  final geoMethods = GeoMethods(
    /// [Get API key](https://developers.google.com/maps/documentation/embed/get-
api-key)
    googleApiKey: 'AIzaSyCQJzzJppTzu5uyFv0sutYCwqVFWtu9_OM',
    language: 'es-419',
```

```
    countryCode: 'ec',
  );

  bool isStart = true;
  bool isStopped = false;
  bool isEnded = false;

  @override
  Widget build(BuildContext context) {
    List<Widget> pages = <Widget>[
      const SizedBox.shrink(),
      !isEnded && !isStopped && isStart ? const TripDataScreen() : const
DynamicTripDataScreen(),
      const ScoreScreen(),
      const TripHistoryScreen(),
    ];
    return Scaffold(
      appBar: AppBar(
        centerTitle: true,
        title: const Text('Driver Behavior Classification'),
        backgroundColor: Colors.purpleAccent,
      ),
      body: _selectedIndex != 0
          ? pages.elementAt(_selectedIndex)
          : Stack(
              children: [
                GoogleMap(
                  myLocationEnabled: true,
                  markers: markers,
                  polylines: Set<Polyline>.of(polyline),
                  onMapCreated: (GoogleMapController controller) async {
                    googleMapsController.complete(controller);
                  },
                  onLongPress: (coords) async {
                    await positionCameraToRoute(polyline);
                  },
                  initialCameraPosition:
                      const CameraPosition(target: LatLng(49.835063, 24.011774),
zoom: 15),
                ),
                ElevatedButton(
                  onPressed: () {
                    final lat = detail.result.geometry?.location.lat;
                    final lng = detail.result.geometry?.location.lng;

                    if (lat != null && lng != null) {
                      destinationPoint = Location(lat: lat, lng: lng);
                      markers.add(Marker(
                          markerId: const MarkerId("destination"),
                          position: LatLng(destinationPoint!.lat,
destinationPoint!.lng)));
                      drawRoute(LatLng(destinationPoint!.lat,
destinationPoint!.lng));
                    }
                  },
                  child: Text("Search places"),
                ),

                if (isStart)
                  Align(
                    alignment: Alignment.bottomCenter,
```

```
child: ElevatedButton(
  onPressed: () async {
    await startTracking();
    setState(() {
      isStart = false;
      isStopped = true;
      isEnded = false;
    });

    const snackBarDanger = SnackBar(
      content: Row(
        children: [
          Icon(Icons.dangerous_outlined, color: Colors.white),
          SizedBox(width: 10),
          Expanded(
            child: Text(
              'Sharp braking!',
              style: TextStyle(
                fontSize: 16,
                fontWeight: FontWeight.w500,
              ),
            ),
          ),
        ],
      ),
      backgroundColor: Colors.red,
      duration: Duration(seconds: 4),
      elevation: 2,
      showCloseIcon: true,
      behavior: SnackBarBehavior.floating,
    );

    const snackBarWarning = SnackBar(
      content: Row(
        children: [
          Icon(Icons.info_outline, color: Colors.white),
          SizedBox(width: 10),
          Expanded(
            child: Text(
              'Sudden left turn!',
              style: TextStyle(
                fontSize: 16,
                fontWeight: FontWeight.w500,
              ),
            ),
          ),
        ],
      ),
      backgroundColor: Colors.amber,
      duration: Duration(seconds: 4),
      elevation: 2,
      showCloseIcon: true,
      behavior: SnackBarBehavior.floating,
    );

    const snackBarWellDone = SnackBar(
      content: Row(
        children: [
          Icon(Icons.check_circle, color: Colors.white),
          SizedBox(width: 10),
          Expanded(
```

```
                                child: Text(
                                  'Well done!',
                                  style: TextStyle(
                                    fontSize: 16,
                                    fontWeight: FontWeight.w500,
                                  ),
                                ),
                              ),
                            ],
                          ),
                          backgroundColor: Colors.green,
                          showCloseIcon: true,
                          duration: Duration(seconds: 4),
                          elevation: 2,
                          behavior: SnackBarBehavior.floating,
                        );
{
                        await Future.delayed(Duration(seconds: 7));


ScaffoldMessenger.of(context).showSnackBar(snackBarDanger);

                        await Future.delayed(Duration(seconds: 8));

ScaffoldMessenger.of(context).showSnackBar(snackBarWarning);
                        await Future.delayed(Duration(seconds: 9));

ScaffoldMessenger.of(context).showSnackBar(snackBarWarning);
                        await Future.delayed(Duration(seconds: 5));

ScaffoldMessenger.of(context).showSnackBar(snackBarWellDone);
                      },
                      child: const Text('Start Tracking'),
                    ),
                  ),
                if (isStopped)
                  Align(
                    alignment: Alignment.bottomCenter,
                    child: ElevatedButton(
                      onPressed: () async {
                        await stopTracking();
                        setState(() {
                          isStart = false;
                          isStopped = false;
                          isEnded = true;
                        });

                        if (!mounted) return;
                        ScaffoldMessenger.of(context).clearSnackBars();
                      },
                      child: const Text('Stop Tracking'),
                    ),
                  ),
                if (isEnded)
                  Align(
                    alignment: Alignment.bottomCenter,
                    child: ElevatedButton(
                      onPressed: () async {
                        await _showAlertDialog();
                        setState(() {
                          isStart = true;
```

```
                        isStopped = false;
                        isEnded = false;
                        polyline.clear();
                      });
                    },
                    child: const Text('End ride'),
                  ),
                ),

            ],
          ),
      bottomNavigationBar: BottomNavigationBar(
        backgroundColor: Colors.purpleAccent,
        selectedItemColor: Colors.black,
        unselectedItemColor: Colors.black38,
        currentIndex: _selectedIndex,
        onTap: _onItemTapped,
        items: const <BottomNavigationBarItem>[
          BottomNavigationBarItem(
            icon: Icon(Icons.map_rounded),
            label: 'Map',
          ),
          BottomNavigationBarItem(
            icon: Icon(Icons.notes_outlined),
            label: 'Trip Data',
          ),
          BottomNavigationBarItem(
            icon: Icon(Icons.score_outlined),
            label: 'Score',
          ),
          BottomNavigationBarItem(
            icon: Icon(Icons.history),
            label: 'Trip History',
          ),
        ],
      ),
    );
}

Future<void> _showAlertDialog() async {
  return showDialog<void>(
    context: context,
    barrierDismissible: false,
    builder: (BuildContext context) {
      return AlertDialog(
        title: const Text('Your ride is finished!'),
        actions: <Widget>[
          TextButton(
            child: const Text('Ok'),
            onPressed: () {
              Navigator.of(context).pop();
            },
          ),
        ],
      );
    },
  );
}

Set<Polyline> polyline = <Polyline>{};
```

```
  Future<PolylineModel> fetchPolylinePoints(
      LatLng origin, LatLng destination, PolylinePoints polylinePoints) async {
    try {
      PolylineResult result = await polylinePoints.getRouteBetweenCoordinates(
        apiKey,
        PointLatLng(origin.latitude, origin.longitude),
        PointLatLng(destination.latitude, destination.longitude),
      );
      List<LatLng> polylineCoordinatesLocal = [];
      Set<Polyline> polyline = {};
      if (result.status == 'OK') {
        for (var point in result.points) {
          polylineCoordinatesLocal.add(LatLng(point.latitude, point.longitude));
        }
      } else {
        return Future.error('API request failed with status: ${result.status}');
      }
      polyline = _drawPolyline(polylineCoordinatesLocal).polyline;
      return PolylineModel(polylineCoordinates: polylineCoordinatesLocal,
polyline: polyline);
    } catch (error) {
      log('Error getRouteBetweenCoordinates: $error');
      return PolylineModel(polylineCoordinates: [], polyline: {});
    }
  }

  PolylineModel _drawPolyline(List<LatLng> polylineCoordinates) {
    Set<Polyline> polylines = {};
    Polyline polyline = Polyline(
      polylineId: const PolylineId('polyline'),
      color: Colors.blue,
      points: polylineCoordinates,
      width: 5,
    );
    polylines.add(polyline);
    setState(() {});
    return PolylineModel(polylineCoordinates: [], polyline: polylines);
  }

  void drawRoute(LatLng destination) async {
    try {
      PolylinePoints polylinePoints = PolylinePoints();
      if (polyline.isNotEmpty) {
      }
      bg.Location myCurrentLocation = await _getCurrentLocation();
      await fetchPolylinePoints(
              LatLng(myCurrentLocation.coords.latitude,
myCurrentLocation.coords.longitude),
              LatLng(destination.latitude, destination.longitude),
              polylinePoints)
          .then((value) {
        return polyline.add(value.polyline.first);
      }).onError((error, stackTrace) {
        log('Error drawRoute $error');
        return false;
      });
    }
  }

  static const apiKey = 'AIzaSyCQJzzJppTzu5uyFv0sutYCwqVFWtu9_OM';
```

```
  Future<void> positionCameraToRoute(Set<Polyline> polylines) async {
    try {
      double minLat = polylines.first.points.first.latitude;
      double minLong = polylines.first.points.first.longitude;
      double maxLat = polylines.first.points.first.latitude;
      double maxLong = polylines.first.points.first.longitude;
      for (var poly in polylines) {
        for (var point in poly.points) {
          if (point.latitude < minLat) minLat = point.latitude;
          if (point.latitude > maxLat) maxLat = point.latitude;
          if (point.longitude < minLong) minLong = point.longitude;
          if (point.longitude > maxLong) maxLong = point.longitude;
        }
      }
      var c = await googleMapsController.future;
      c.animateCamera(CameraUpdate.newLatLngBounds(
          LatLngBounds(southwest: LatLng(minLat, minLong), northeast:
LatLng(maxLat, maxLong)),
          50));

    } catch (e) {}
  }

  int _selectedIndex = 0;

  void _onItemTapped(int index) {
    setState(() {
      _selectedIndex = index;
    });
  }

  Future<void> getReady() async {
    // try {
    await bg.BackgroundGeolocation.ready(bg.Config(
            desiredAccuracy: bg.Config.DESIRED_ACCURACY_HIGH,
            distanceFilter: 10.0,
            stopOnTerminate: false,
            startOnBoot: true,
            debug: true,
            logLevel: bg.Config.LOG_LEVEL_VERBOSE))
        .then((state) {
      if (!state.enabled) {
      }
    });

    bg.BackgroundGeolocation.onLocation((location) async {
      if (location.sample == false) {
        print('*** LOCATION $location');
        if (destinationPoint != null) {
          drawRoute(LatLng(destinationPoint!.lat, destinationPoint!.lng));
      }
    });
  }


  Future<void> _handlePressButton() async {

    final p = await PlacesAutocomplete.show(
      context: context,
      apiKey: apiKey,
      onError: (error) {
```

```
        print(error.errorMessage);
      },
      mode: Mode.overlay,
      language: "en",
      types: [],
      strictbounds: false,
      decoration: InputDecoration(
        hintText: 'Search',
        focusedBorder: OutlineInputBorder(
          borderRadius: BorderRadius.circular(20),
          borderSide: const BorderSide(
            color: Colors.white,
          ),
        ),
      ),
      components: [Component(Component.country, "ua")],
    );
    displayPrediction(p!);
  }

  Future<void> displayPrediction(Prediction p) async {
    GoogleMapsPlaces _places = GoogleMapsPlaces(
      apiKey: apiKey,
      apiHeaders: await GoogleApiHeaders().getHeaders(),
    );
    PlacesDetailsResponse detail = await _places.getDetailsByPlaceId(p.placeId ??
'');
    final lat = detail.result.geometry?.location.lat;
    final lng = detail.result.geometry?.location.lng;

    if (lat != null && lng != null) {
      destinationPoint = Location(lat: lat, lng: lng);
      markers.add(Marker(
          markerId: const MarkerId("destination"),
          position: LatLng(destinationPoint!.lat, destinationPoint!.lng)));
      drawRoute(LatLng(destinationPoint!.lat, destinationPoint!.lng));
    }

    setState(() {});

    if (!mounted) return;
    ScaffoldMessenger.of(context)
        .showSnackBar(SnackBar(content: Text("${p.description} - $lat/$lng")));
  }

  Future<void> startTracking() async {
    await bg.BackgroundGeolocation.start();
  }

  Future<void> stopTracking() async {
    await bg.BackgroundGeolocation.stop();
  }

  Future<bg.Location> _getCurrentLocation() =>
      bg.BackgroundGeolocation.getCurrentPosition(samples: 1);
}

class RatingWidget extends StatelessWidget {
  const RatingWidget({super.key, required this.initialRating, required
this.characteristics});
```

```
  final double initialRating;
  final String characteristics;

  @override
  Widget build(BuildContext context) {
    return Row(
      mainAxisAlignment: MainAxisAlignment.spaceBetween,
      children: [
        Text(
          characteristics,
          style: const TextStyle(
            fontSize: 16,
          ),
        ),
        Padding(
          padding: const EdgeInsets.only(right: 30),
          child: RatingBar.builder(
            initialRating: initialRating,
            itemCount: 5,
            itemPadding: const EdgeInsets.only(left: 5),
            itemBuilder: (context, index) {
              switch (index) {
                case 0:
                  return const Icon(
                    Icons.sentiment_very_dissatisfied,
                    color: Colors.red,
                  );
                case 1:
                  return const Icon(
                    Icons.sentiment_dissatisfied,
                    color: Colors.redAccent,
                  );
                case 2:
                  return const Icon(
                    Icons.sentiment_neutral,
                    color: Colors.amber,
                  );
                case 3:
                  return const Icon(
                    Icons.sentiment_satisfied,
                    color: Colors.lightGreen,
                  );
                case 4:
                  return const Icon(
                    Icons.sentiment_very_satisfied,
                    color: Colors.green,
                  );
              }
              return const SizedBox.shrink();
            },
            onRatingUpdate: (rating) {
            },
          ),
        ),
      ],
    );
  }
}

class ScoreScreen extends StatefulWidget {
  const ScoreScreen({super.key});
```

```
  @override
  State<ScoreScreen> createState() => _ScoreScreenState();
}

class _ScoreScreenState extends State<ScoreScreen> {
  @override
  Widget build(BuildContext context) {
    return SingleChildScrollView(
      child: Column(
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
          const SizedBox(height: 16),
          DashedCircularProgressBar.aspectRatio(
            aspectRatio: 1.5,
            progress: score,
            startAngle: 225,
            sweepAngle: 270,
            foregroundColor: Colors.green,
            backgroundColor: Color(0xffeeeeee),
            foregroundStrokeWidth: 15,
            backgroundStrokeWidth: 15,
            animation: true,
            seekSize: 6,
            seekColor: Color(0xffeeeeee),
            child: Center(
              child: Column(
                mainAxisSize: MainAxisSize.min,
                children: [
                  Text(
                    '${score.toStringAsFixed(0)}%',
                    style: TextStyle(color: Colors.black, fontSize: 40),
                  ),
                  SizedBox(height: 30),
                  Text(
                    'Score',
                    style: TextStyle(color: Colors.black, fontSize: 30),
                  ),
                ],
              ),
            ),
          ),
          Row(
            mainAxisAlignment: MainAxisAlignment.spaceEvenly,
            children: [
              Container(
                width: 80,
                //padding: const EdgeInsets.symmetric(horizontal: 10),
                decoration: BoxDecoration(
                  border: Border.all(),
                  borderRadius: BorderRadius.circular(10),
                ),
                child: Column(
                  children: [
                    Text(
                      trips.toString(),
                      style: TextStyle(
                        fontWeight: FontWeight.w700,
                        fontSize: 25,
                      ),
                    ),
```

```
          Text('Trips')
        ],
      ),
    ),
    Container(
      width: 80,
      // padding: const EdgeInsets.symmetric(horizontal: 10),
      decoration: BoxDecoration(
        border: Border.all(),
        borderRadius: BorderRadius.circular(10),
      ),
      child: Column(
        children: [
          Text(
            kilometers.toString(),
            style: TextStyle(
              fontWeight: FontWeight.w700,
              fontSize: 25,
            ),
          ),
          Text('Kilometers')
        ],
      ),
    ),
    Container(
      width: 80,
      //padding: const EdgeInsets.symmetric(horizontal: 10),
      decoration: BoxDecoration(
        border: Border.all(),
        borderRadius: BorderRadius.circular(10),
      ),
      child: Column(
        children: [
          Text(
            hours.toString(),
            style: TextStyle(
              fontWeight: FontWeight.w700,
              fontSize: 25,
            ),
          ),
          Text('Hours')
        ],
      ),
    ),
  ],
),
SizedBox(height: 8),
Center(
  child: Text(
    'Your overal points is 1228!',
    style: TextStyle(
      fontWeight: FontWeight.w500,
      fontSize: 20,
    ),
  ),
),
const Divider(),
Padding(
  padding: const EdgeInsets.symmetric(horizontal: 16),
  child: Column(
    crossAxisAlignment: CrossAxisAlignment.start,
```

```
              children: [
                SizedBox(height: 8),
                Text(
                  'Challengies',
                  style: TextStyle(
                    fontWeight: FontWeight.w500,
                    fontSize: 20,
                  ),
                ),
                SizedBox(height: 16),
                Text(
                  'Drive 100 km with safely speed 35+',
                  style: TextStyle(
                    fontSize: 18,
                  ),
                ),
                Text(
                  'Drive 2 weeks without sharp braking 120+',
                  style: TextStyle(
                    fontSize: 18,
                  ),
                ),
                Text(
                  'Drive 1 week with safely cornering 90+',
                  style: TextStyle(
                    fontSize: 18,
                  ),
                ),
                Text(
                  'Drive 120 km and don\'t over accelerate 75+',
                  style: TextStyle(
                    fontSize: 18,
                  ),
                ),
              ],
            ),
          ),
        ],
      ),
    );
  }
}

class EntityExtractionView extends StatefulWidget {
  @override
  State<EntityExtractionView> createState() => _EntityExtractionViewState();
}

class _EntityExtractionViewState extends State<EntityExtractionView> {
  final _controller = TextEditingController();
  final _modelManager = EntityExtractorModelManager();
  final _entityExtractor =
  EntityExtractor(language: EntityExtractorLanguage.english);
  var _entities = <EntityAnnotation>[];
  final _language = EntityExtractorLanguage.english;

  @override
  void dispose() {
    _entityExtractor.close();
    super.dispose();
  }
```

```
@override
Widget build(BuildContext context) {
  return SafeArea(
    child: Scaffold(
      appBar: AppBar(
        title: const Text('Entity Extractor'),
      ),
      body: GestureDetector(
        onTap: () {
          FocusScope.of(context).unfocus();
        },
        child: SingleChildScrollView(
          child: Column(
            children: [
              const SizedBox(
                height: 20,
              ),
              const Center(child: Text('Enter text (English)')),
              Padding(
                padding: const EdgeInsets.all(20.0),
                child: Container(
                  padding: EdgeInsets.symmetric(horizontal: 20),
                  decoration: BoxDecoration(
                      border: Border.all(
                        width: 2,
                      )),
                  child: TextField(
                    controller: _controller,
                    decoration: InputDecoration(border: InputBorder.none),
                    maxLines: null,
                  ),
                ),
              ),
              Row(mainAxisAlignment: MainAxisAlignment.center, children: [
                ElevatedButton(
                    onPressed: _extractEntities,
                    child: Text('Extract Entities'))
              ]),
              const SizedBox(height: 20),
              Row(
                mainAxisAlignment: MainAxisAlignment.spaceAround,
                children: [
                  ElevatedButton(
                      onPressed: _downloadModel,
                      child: Text('Download Model')),
                  ElevatedButton(
                      onPressed: _deleteModel, child: Text('Delete Model')),
                ],
              ),
              SizedBox(height: 20),
              Row(
                  mainAxisAlignment: MainAxisAlignment.spaceAround,
                  children: [
                    ElevatedButton(
                        onPressed: _isModelDownloaded,
                        child: Text('Check download'))
                  ]),
              const SizedBox(
                height: 30,
              ),
```

```
                Container(
                  padding: const EdgeInsets.symmetric(horizontal: 30),
                  child: const Text('Result', style: TextStyle(fontSize: 20)),
                ),
                ListView.builder(
                  padding: const EdgeInsets.symmetric(horizontal: 10),
                  shrinkWrap: true,
                  physics: NeverScrollableScrollPhysics(),
                  itemCount: _entities.length,
                  itemBuilder: (context, index) => ExpansionTile(
                      title: Text(_entities[index].text),
                      children: _entities[index]
                          .entities
                          .map((e) => Text(e.toString()))
                          .toList()),
                ),
              ],
            ),
          ),
        ),
      ),
    );
  }

  Future<void> _downloadModel() async {
    SnackBar(
        content: Text('Downloading model...'),
        onVisible: (){_modelManager
        .downloadModel(_language.name)
        .then((value) => value ? 'success' : 'failed');
    },);
  }

  Future<void> _deleteModel() async {
    SnackBar(
        content: Text('Deleting model...'),
        onVisible: () {
          _modelManager
              .deleteModel(_language.name)
              .then((value) => value ? 'success' : 'failed');
        },);
  }

  Future<void> _isModelDownloaded() async {
    SnackBar(
        content: Text('Checking if model is downloaded...'),
        onVisible:() {
          _modelManager
              .isModelDownloaded(_language.name)
              .then((value) => value ? 'downloaded' : 'not downloaded');
        });
  }

  Future<void> _extractEntities() async {
    FocusScope.of(context).unfocus();
    final result = await _entityExtractor.annotateText(_controller.text);
    setState(() {
      _entities = result;
    });
  }
}
```

```
class NormalProgressIndicator extends StatelessWidget {
  const NormalProgressIndicator({super.key, required this.value});

  final double value;

  @override
  Widget build(BuildContext context) {
    return SizedBox(
      height: 150,
      width: 130,
      child: SfRadialGauge(
        axes: [
          RadialAxis(
            minimum: 0,
            maximum: 100,
            showLabels: false,
            showTicks: false,
            radiusFactor: 0.8,
            axisLineStyle: const AxisLineStyle(
              thickness: 0.2,
              cornerStyle: CornerStyle.bothCurve,
              color: Colors.lightBlueAccent,
              thicknessUnit: GaugeSizeUnit.factor,
            ),
            pointers: <GaugePointer>[
              RangePointer(
                value: progressValue,
                cornerStyle: CornerStyle.bothCurve,
                color: Colors.purple,
                width: 0.2,
                sizeUnit: GaugeSizeUnit.factor,
                enableAnimation: true,
                animationDuration: 100,
                animationType: AnimationType.linear,
              ),
            ],
            annotations: [
              GaugeAnnotation(
                positionFactor: 0.1,
                angle: 90,
                widget: Text(
                  '${(value / 100).toStringAsFixed(2)} °/s',
                  style: const TextStyle(fontSize: 11),
                ),
              ),
            ],
          )
        ],
      ),
    );
  }
}


class RoundedProgressIndicator extends StatelessWidget {
  const RoundedProgressIndicator({super.key});

  @override
  Widget build(BuildContext context) {
    return SizedBox(
```

```
      height: 120,
      width: 120,
      child: SfRadialGauge(
        axes: [
          RadialAxis(
            minimum: 0,
            maximum: 100,
            showLabels: false,
            showTicks: false,
            startAngle: 270,
            endAngle: 270,
            radiusFactor: 0.8,
            axisLineStyle: const AxisLineStyle(
              thickness: 0.05,
              color: Colors.blue,
              thicknessUnit: GaugeSizeUnit.factor,
            ),
            pointers: const <GaugePointer>[
              RangePointer(
                value: progressValue,
                width: 0.05,
                sizeUnit: GaugeSizeUnit.factor,
                enableAnimation: true,
                animationDuration: 100,
                animationType: AnimationType.linear,
                color: Colors.purple,
              )
            ],
            annotations: const [GaugeAnnotation(positionFactor: 0, widget:
Text('80 %'))],
          ),
        ],
      ),
    );
  }
}

class SegmentedProgressIndicator extends StatelessWidget {
  const SegmentedProgressIndicator({super.key, required this.value});

  final double value;

  @override
  Widget build(BuildContext context) {
    return SizedBox(
      height: 150,
      width: 130,
      child: SfRadialGauge(
        axes: [
          RadialAxis(
            minimum: 0,
            maximum: 100,
            showLabels: false,
            showTicks: false,
            radiusFactor: 0.8,
            axisLineStyle: const AxisLineStyle(
              thickness: 0.3,
              color: Colors.lightBlueAccent,
              thicknessUnit: GaugeSizeUnit.factor,
            ),
            pointers: [
```

```
              RangePointer(
                  value: progressValue,
                  width: 0.3,
                  color: Colors.purple,
                  sizeUnit: GaugeSizeUnit.factor,
                  enableAnimation: true,
                  animationDuration: 100,
                  animationType: AnimationType.linear)
            ],
            annotations: [
              GaugeAnnotation(
                  angle: 90,
                  positionFactor: 0.2,
                widget: Text('${(value/100).toStringAsFixed(2)}\nm/s^2'),
              )
            ],
          ),

          RadialAxis(
            minimum: 0,
            maximum: 100,
            showLabels: false,
            showTicks: false,
            showAxisLine: true,
            tickOffset: -0.05,
            offsetUnit: GaugeSizeUnit.factor,
            minorTicksPerInterval: 0,
            radiusFactor: 0.8,
            axisLineStyle: const AxisLineStyle(
              thickness: 0.3,
              color: Colors.white,
              dashArray: <double>[4, 3],
              thicknessUnit: GaugeSizeUnit.factor,
            ),
          )
        ],
      ),
    );
  }
}


class DynamicTripDataScreen extends StatefulWidget {
  const DynamicTripDataScreen({super.key});

  @override
  State<DynamicTripDataScreen> createState() => _DynamicTripDataScreenState();
}

class _DynamicTripDataScreenState extends State<DynamicTripDataScreen> {


  @override
  Widget build(BuildContext context) {
    return SingleChildScrollView(
      child: Column(
        children: [
          const SizedBox(height: 16),
          Column(
            children: [
              SizedBox(
```

```
                height: 250,
                child: SfRadialGauge(
                  axes: <RadialAxis>[
                    RadialAxis(minimum: 0, maximum: 150, ranges: <GaugeRange>[
                      GaugeRange(startValue: 0, endValue: 50, color:
Colors.green),
                      GaugeRange(startValue: 50, endValue: 100, color:
Colors.orange),
                      GaugeRange(startValue: 100, endValue: 150, color:
Colors.red)
                    ], pointers: [
                      NeedlePointer(value: speed)
                    ], annotations: [
                      GaugeAnnotation(
                          widget: Text(' ${speed.toStringAsFixed(0)}',
                              style: TextStyle(fontSize: 25, fontWeight:
FontWeight.bold)),
                          angle: 90,
                          positionFactor: 0.5),
                    ]),
                  ],
                ),
              ),
              const Text(
                'speed',
                style: TextStyle(
                  height: -2.5,
                  fontSize: 20,
                ),
              ),
            ],
          ),
          Row(
            mainAxisAlignment: MainAxisAlignment.spaceEvenly,
            children: [
              Column(
                children: [
                  SegmentedProgressIndicator(
                    value: accX,
                  ),
                  Text(
                    'acceleration X',
                    style: TextStyle(
                      height: -1.5,
                    ),
                  ),
                ],
              ),
              Column(
                children: [
                  SegmentedProgressIndicator(
                    value: accY,
                  ),
                  Text(
                    'acceleration Y',
                    style: TextStyle(
                      height: -1.5,
                    ),
                  ),
                ],
              ),
```

```
      Column(
        children: [
          SegmentedProgressIndicator(
            value: accZ,
          ),
          Text(
            'acceleration Z',
            style: TextStyle(
              height: -1.5,
            ),
          ),
        ],
      ),
    ],
  ),
  Row(
    mainAxisAlignment: MainAxisAlignment.spaceAround,
    children: [
      Column(
        children: [
          NormalProgressIndicator(
            value: rotationX,
          ),
          Text(
            'rotation X',
            style: TextStyle(
              height: -1.5,
            ),
          ),
        ],
      ),
      Column(
        children: [
          NormalProgressIndicator(
            value: rotationY,
          ),
          Text(
            'rotation Y',
            style: TextStyle(
              height: -1.5,
            ),
          ),
        ],
      ),
      Column(
        children: [
          NormalProgressIndicator(
            value: rotationZ,
          ),
          Text(
            'rotation Z',
            style: TextStyle(
              height: -1.5,
            ),
          ),
        ],
      ),
    ],
  ),

],
```

```
      ),
    );
  }
}

class _NodesMarkupData {
  _NodesMarkupData(
      {required this.markup, required this.level, required this.y});

  final String markup;
  final int level;
  final num y;
}

String createTreeSvgMarkup(TreeNode node) {
  final shape = node.shape;
  final levels = getTreeLevels(node, shape.length);
  final nodeDistanceByLevel =
      getTreeNodeDistanceByLevel(levels, nodeWidth, minNodeHorizontalDistance);
  final totalWidth = getTreeWidth(levels, nodeWidth, minNodeHorizontalDistance);
  final totalHeight = shape.length * (nodeHeight + nodeVerticalDistance);
  final markup = _generateMarkup(levels, node, nodeDistanceByLevel);

  return '<svg xmlns="http://www.w3.org/2000/svg" width="$totalWidth"
height="$totalHeight">$textStyles$markup</svg>';
}

String _generateMarkup(
    List<List<TreeNode>> levels, TreeNode root, Map<int, num> distByLevel) {
  return levels.fold<_NodesMarkupData>(
      _NodesMarkupData(markup: '', level: 0, y: 20), (data, nodes) {
    final spacing = distByLevel[data.level]!;
    final childSpacing = distByLevel.containsKey(data.level + 1)
        ? distByLevel[data.level + 1]!
        : null;
    final getX =
        (int idx) => spacing / 2 + (idx == 0 ? 0 : idx * (nodeWidth + spacing));

    var nodeIdx = 0;
    final nodesMarkup = nodes
        .map((node) =>
            getTreeNodeMarkup(node, getX(nodeIdx++), data.y, childSpacing))
        .join();

    return _NodesMarkupData(
      markup: '${data.markup}$nodesMarkup',
      level: data.level + 1,
      y: data.y + nodeHeight + nodeVerticalDistance,
    );
  }).markup;
}

const nodeWidth = 200;
const nodeHeight = 80;
const minNodeHorizontalDistance = 20;
const nodeVerticalDistance = 150;

const textStyles = '<style>'
    '.label { font: lighter 16px sans-serif; fill: #fff }'
    '.value { font: bold 16px sans-serif; fill: #fff }'
    '.root-node-label { font: 24px sans-serif; fill: #fff }'
```

```
    '</style>';

const nodeStyle = 'fill:purple;fill-opacity:.6;stroke:purple;stroke-width:3';
const nodeLineStyle = 'stroke:purple;stroke-width:2';

const labelMargin = 20;
const labelSpacing = 10;
const labelWidth = 100;
const labelHeight = 20;
const noValue = '-';

class _LeveledNode {
  _LeveledNode(this.node, this.level);

  final TreeNode node;
  final int level;
}

TreeNode _createFakeNode() {
  return TreeNode(null, null, null, null, null);
}

List<List<TreeNode>> getTreeLevels(TreeNode node, int depth) {
  var currentLevel = -1;
  final queue = [_LeveledNode(node, 0)];
  final levels = <List<TreeNode>>[];

  while (queue.isNotEmpty) {
    final node = queue.removeAt(0);
    final children = node.node.children?.isNotEmpty == true
        ? node.node.children!
        : node.level + 1 == depth
            ? <TreeNode>[]
            : [_createFakeNode(), _createFakeNode()];

    if (currentLevel != node.level) {
      levels.add([node.node]);
    } else {
      levels[currentLevel].add(node.node);
    }

    currentLevel = node.level;

    queue.addAll(children.map((child) => _LeveledNode(child, node.level + 1)));
  }

  return levels;
}

Map<int, num> getTreeNodeDistanceByLevel<T>(
    List<List<T>> levels, num nodeWidth, num minDist) {
  if (levels.length == 1) {
    return {0: minDist};
  }

  final lastLevelIdx = levels.length - 1;
  final totalWidth = getTreeWidth(levels, nodeWidth, minDist);
  final distByLevel = <int, num>{lastLevelIdx: minDist};

  for (var i = 0; i < levels.length - 1; i++) {
    distByLevel[i] =
```

```
        (totalWidth - (levels[i].length * nodeWidth)) / levels[i].length;
  }

  return distByLevel;
}

String getTreeNodeMarkup(TreeNode node, num x, num y, num? childSpacing) {
  if (node.isFake) {
    return '';
  }

  final labelX = x + labelMargin;

  final splitIndex = node.splitIndex;
  final predicateType = node.predicateType;
  final splitValue = node.splitValue;
  final rectMarkup = _getRectMarkup(x, y);
  final linesMarkup = getTreeNodeLinesMarkup(node.children, x, y, childSpacing);

  if (splitIndex == null || predicateType == null || splitValue == null) {
    return _getRootNodeMarkup(x, y, linesMarkup);
  }

  final conditionLabel = formatPredicate(predicateType);
  final valueLabel = formatValue(splitValue);
  final getLabelY = (int num) => y + 15 + labelHeight * num;

  final splitIndexMarkup =
      '<text class="label" x="$labelX" y="${getLabelY(1)}">Column index</text>'
      '<text class="value" x="${labelX + labelWidth + labelSpacing}"
y="${getLabelY(1)}">${node.splitIndex}</text>';

  final predicateMarkup =
      '<text class="label" x="$labelX" y="${getLabelY(2)}">Split condition</text>'
      '<text class="value" x="${labelX + labelWidth + labelSpacing}"
y="${getLabelY(2)}">$conditionLabel $valueLabel</text>';

  return '<g>'
      '$rectMarkup'
      '$splitIndexMarkup'
      '$predicateMarkup'
      '$linesMarkup'
      '</g>';
}

String _getRootNodeMarkup(num x, num y, String linesMarkup) {
  final rectMarkup = _getRectMarkup(x, y);

  return '<g>'
      '$rectMarkup'
      '<text class="root-node-label" x="${x + 65}" y="${y + 50}">ROOT</text>'
      '$linesMarkup'
      '</g>';
}

String _getRectMarkup(num x, num y) {
  return '<rect rx="15" style="$nodeStyle" x="$x" y="$y" width="$nodeWidth"
height="$nodeHeight"></rect>';
}

String formatValue(num value) {
```

```
  return value.toStringAsFixed(2);
}

num getTreeWidth<T>(List<List<T>> levels, num nodeWidth, num minDist) {
  final lastLevel = levels[levels.length - 1];

  return lastLevel.length * (nodeWidth + minDist);
}

mixin LinearClassifierMixin implements LinearClassifier {
  @override
  DataFrame predictProbabilities(DataFrame testFeatures) {
    final probabilities = getProbabilitiesMatrix(testFeatures);

    return DataFrame.fromMatrix(
      probabilities,
      header: targetNames,
    );
  }

  Matrix getProbabilitiesMatrix(DataFrame testFeatures) {
    validateTestFeatures(testFeatures, dtype);

    final processedFeatures = addInterceptIf(
      fitIntercept,
      testFeatures.toMatrix(dtype),
      interceptScale,
      dtype,
    );

    validateCoefficientsMatrix(
        coefficientsByClasses, processedFeatures.columnCount);

    return linkFunction.link(processedFeatures * coefficientsByClasses);
  }
}

LinearOptimizer createLogLikelihoodOptimizer(
  DataFrame fittingData,
  Iterable<String> targetNames,
  LinkFunction linkFunction, {
  required LinearOptimizerType optimizerType,
  required int iterationsLimit,
  required double initialLearningRate,
  required int dropRate,
  required double decay,
  required double minCoefficientsUpdate,
  required double lambda,
  required int batchSize,
  required bool fitIntercept,
  required double interceptScale,
  required bool isFittingDataNormalized,
  required LearningRateType learningRateType,
  required InitialCoefficientsType initialCoefficientsType,
  required num positiveLabel,
  required num negativeLabel,
  required DType dtype,
  RegularizationType? regularizationType,
  int? randomSeed,
}) {
  validateClassLabels(positiveLabel, negativeLabel);
```

```
  final splits =
      featuresTargetSplit(fittingData, targetNames: targetNames).toList();
  final points = splits[0].toMatrix(dtype);
  final labels = splits[1].toMatrix(dtype);
  final optimizerFactory = injector.get<LinearOptimizerFactory>();
  final costFunctionFactory = injector.get<CostFunctionFactory>();
  final costFunction = costFunctionFactory.createByType(
    CostFunctionType.logLikelihood,
    linkFunction: linkFunction,
    positiveLabel: positiveLabel,
    negativeLabel: negativeLabel,
    dtype: dtype,
  );
  final normalizedLabels =
      normalizeClassLabels(labels, positiveLabel, negativeLabel);

  return optimizerFactory.createByType(
    optimizerType,
    addInterceptIf(fitIntercept, points, interceptScale, dtype),
    normalizedLabels,
    costFunction: costFunction,
    iterationLimit: iterationsLimit,
    initialLearningRate: initialLearningRate,
    decay: decay,
    dropRate: dropRate,
    minCoefficientsUpdate: minCoefficientsUpdate,
    lambda: lambda,
    regularizationType: regularizationType,
    randomSeed: randomSeed,
    batchSize: batchSize,
    learningRateType: learningRateType,
    initialCoefficientsType: initialCoefficientsType,
    dtype: dtype,
    isFittingDataNormalized: isFittingDataNormalized,
  );
}

@JsonSerializable()
class TreeNode {
  TreeNode(this.predicateType, this.splitValue, this.splitIndex, this.children,
      this.label);

  factory TreeNode.fromJson(Map<String, dynamic> json) {
    final migrated = migrateTreeNodeJsonSchema(json);

    return _$TreeNodeFromJson(migrated);
  }

  Map<String, dynamic> toJson() => _$TreeNodeToJson(this);

  @JsonKey(
    name: childrenJsonKey,
    toJson: treeNodesToJson,
    fromJson: fromTreeNodesJson,
  )
  final List<TreeNode>? children;

  @JsonKey(name: labelJsonKey)
  final TreeLeafLabel? label;
```

```
  @JsonKey(
    name: predicateTypeJsonKey,
    toJson: predicateTypeToJson,
    fromJson: fromPredicateTypeJson,
  )
  final PredicateType? predicateType;

  @JsonKey(name: splitValueJsonKey)
  final num? splitValue;

  @JsonKey(name: splitIndexJsonKey)
  final int? splitIndex;

  bool get isLeaf => children == null || children!.isEmpty;

  bool get isRoot =>
      predicateType == null || splitIndex == null || splitValue == null;

  bool get isFake =>
      predicateType == null &&
      splitIndex == null &&
      splitValue == null &&
      label == null &&
      children == null;

  bool testSample(Vector sample) {
    if (isRoot) {
      return true;
    }

    final predicate = getSplitPredicateByType(predicateType!);

    return predicate(
      sample,
      splitIndex!,
      splitValue!,
    );
  }

  /// Returns a map where a key is a tree level number, and the value is a
  /// number of nodes on the level
  Map<int, int> get shape {
    final _shape = <int, int>{};

    _collectShape(this, _shape, 0);

    return _shape;
  }

  void _collectShape(TreeNode node, Map<int, int> shape, int level) {
    final children = node.children;
    final childCount = children?.length ?? 0;

    shape.update(level, (count) => count + childCount,
        ifAbsent: () => childCount);

    children?.forEach((child) {
      _collectShape(child, shape, level + 1);
    });
  }
}
```

```
TreeNode _$TreeNodeFromJson(Map<String, dynamic> json) {
  return $checkedNew('TreeNode', json, () {
    $checkKeys(json, allowedKeys: const ['CN', 'LB', 'PT', 'SV', 'SI']);
    final val = TreeNode(
      $checkedConvert(json, 'PT', (v) => fromPredicateTypeJson(v as String?)),
      $checkedConvert(json, 'SV', (v) => v as num?),
      $checkedConvert(json, 'SI', (v) => v as int?),
      $checkedConvert(json, 'CN', (v) => fromTreeNodesJson(v as List?)),
      $checkedConvert(
          json,
          'LB',
          (v) => v == null
              ? null
              : TreeLeafLabel.fromJson(v as Map<String, dynamic>)),
    );
    return val;
  }, fieldKeyMap: const {
    'predicateType': 'PT',
    'splitValue': 'SV',
    'splitIndex': 'SI',
    'children': 'CN',
    'label': 'LB'
  });
}

Map<String, dynamic> _$TreeNodeToJson(TreeNode instance) {
  final val = <String, dynamic>{};

  void writeNotNull(String key, dynamic value) {
    if (value != null) {
      val[key] = value;
    }
  }

  writeNotNull('CN', treeNodesToJson(instance.children));
  writeNotNull('LB', instance.label?.toJson());
  writeNotNull('PT', predicateTypeToJson(instance.predicateType));
  writeNotNull('SV', instance.splitValue);
  writeNotNull('SI', instance.splitIndex);
  return val;
}

class DecisionTreeTrainer implements TreeTrainer {
  DecisionTreeTrainer(
    this._featureIndices,
    this._targetIdx,
    this._featureToValues,
    this._leafDetector,
    this._leafLabelFactory,
    this._splitSelector,
  );

  final Iterable<int> _featureIndices;
  final int _targetIdx;
  final Map<int, List<num>> _featureToValues;
  final TreeLeafDetector _leafDetector;
  final TreeLeafLabelFactory _leafLabelFactory;
  final TreeSplitSelector _splitSelector;

  @override
```

```
TreeNode train(Matrix samples) =>
    _train(samples, null, null, null, _featureIndices, 0);

TreeNode _train(
  Matrix samples,
  num? splitValue,
  int? splitIdx,
  PredicateType? predicateType,
  Iterable<int> featuresColumnIdxs,
  int level,
) {
  final isLeaf = _leafDetector.isLeaf(
    samples,
    _targetIdx,
    featuresColumnIdxs,
    level,
  );

  if (isLeaf) {
    final label = _leafLabelFactory.create(
      samples,
      _targetIdx,
    );

    return TreeNode(
      predicateType,
      splitValue,
      splitIdx,
      null,
      label,
    );
  }

  final bestSplit = _splitSelector.select(
    samples,
    _targetIdx,
    featuresColumnIdxs,
    _featureToValues,
  );

  final childNodes = bestSplit.entries.map((entry) {
    final splitNode = entry.key;
    final splitSamples = entry.value;
    final isSplitByNominalValue =
        _featureToValues.containsKey(splitNode.splitIndex);
    final updatedColumnIdxs = isSplitByNominalValue
        ? (Set<int>.from(featuresColumnIdxs)..remove(splitNode.splitIndex))
        : featuresColumnIdxs;

    return _train(
      splitSamples,
      splitNode.splitValue,
      splitNode.splitIndex,
      splitNode.predicateType,
      updatedColumnIdxs,
      level + 1,
    );
  });

  return TreeNode(
    predicateType,
```

```
      splitValue,
      splitIdx,
      childNodes.toList(growable: false),
      null,
    );
  }
}

class TreeTrainerFactoryImpl implements TreeTrainerFactory {
  const TreeTrainerFactoryImpl(
    this._leafDetectorFactory,
    this._leafLabelFactoryFactory,
    this._splitSelectorFactory,
  );

  final TreeLeafDetectorFactory _leafDetectorFactory;
  final TreeLeafLabelFactoryFactory _leafLabelFactoryFactory;
  final TreeSplitSelectorFactory _splitSelectorFactory;

  @override
  TreeTrainer createByType(
    TreeTrainerType type,
    DataFrame samples,
    String targetName,
    num minErrorOnNode,
    int minSamplesCount,
    int maxDepth,
    TreeAssessorType leafAssessorType,
    TreeLeafLabelFactoryType leafLabelFactoryType,
    TreeSplitSelectorType splitSelectorType,
    TreeAssessorType splitAssessorType,
    TreeSplitterType splitterType,
  ) {
    final targetIdx = enumerate(samples.header)
        .firstWhere((indexedName) => indexedName.value == targetName)
        .index;
    final featuresIndexedSeries = enumerate(samples.series)
        .where((indexed) => indexed.index != targetIdx);
    final featureIdxToUniqueValues = Map.fromEntries(
      featuresIndexedSeries.where((indexed) => indexed.value.isDiscrete).map(
          (indexed) => MapEntry(
              indexed.index,
              indexed.value.discreteValues
                  .map((dynamic value) => value as num)
                  .toList(growable: false))),
    );
    final leafDetector = _leafDetectorFactory.create(
        leafAssessorType, minErrorOnNode, minSamplesCount, maxDepth);

    final leafLabelFactory =
        _leafLabelFactoryFactory.createByType(leafLabelFactoryType);

    final splitSelector = _splitSelectorFactory.createByType(
        splitSelectorType, splitAssessorType, splitterType);

    switch (type) {
      case TreeTrainerType.decision:
        return DecisionTreeTrainer(
          featuresIndexedSeries.map((indexed) => indexed.index),
          targetIdx,
          featureIdxToUniqueValues,
```

```
          leafDetector,
          leafLabelFactory,
          splitSelector,
        );

      default:
        throw UnsupportedError('Tree solver type $type is unsupported');
    }
  }
}

abstract class TreeTrainerFactory {
  TreeTrainer createByType(
    TreeTrainerType type,
    DataFrame samples,
    String targetName,
    num minErrorOnNode,
    int minSamplesCount,
    int maxDepth,
    TreeAssessorType leafAssessorType,
    TreeLeafLabelFactoryType leafLabelFactoryType,
    TreeSplitSelectorType splitSelectorType,
    TreeAssessorType splitAssessorType,
    TreeSplitterType splitterType,
  );
}
```

```
plugins {
    id "com.android.application"
    id "kotlin-android"
    id "dev.flutter.flutter-gradle-plugin"
}

def localProperties = new Properties()
def localPropertiesFile = rootProject.file('local.properties')
if (localPropertiesFile.exists()) {
    localPropertiesFile.withReader('UTF-8') { reader ->
        localProperties.load(reader)
    }
}

def flutterVersionCode = localProperties.getProperty('flutter.versionCode')
if (flutterVersionCode == null) {
    flutterVersionCode = '1'
}

def flutterVersionName = localProperties.getProperty('flutter.versionName')
if (flutterVersionName == null) {
    flutterVersionName = '1.0'
}

apply plugin: 'com.android.application'
apply plugin: 'kotlin-android'

Project background_geolocation = project(':flutter_background_geolocation')
apply from: "${background_geolocation.projectDir}/background_geolocation.gradle"

android {
    namespace "com.example.driving_patterns"
    compileSdkVersion flutter.compileSdkVersion
    ndkVersion flutter.ndkVersion
```

```
    compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_8
        targetCompatibility JavaVersion.VERSION_1_8
    }

    kotlinOptions {
        jvmTarget = '1.8'
    }

    sourceSets {
        main.java.srcDirs += 'src/main/kotlin'
    }

    defaultConfig {
        // TODO: Specify your own unique Application ID
(https://developer.android.com/studio/build/application-id.html).
        applicationId "com.example.driving_patterns"
        // You can update the following values to match your application needs.
        // For more information, see:
https://docs.flutter.dev/deployment/android#reviewing-the-gradle-build-
configuration.
        minSdkVersion 21
        multiDexEnabled true
        targetSdkVersion flutter.targetSdkVersion
        versionCode flutterVersionCode.toInteger()
        versionName flutterVersionName
    }

    buildTypes {
        release {
            // TODO: Add your own signing config for the release build.
            // Signing with the debug keys for now, so `flutter run --release`
works.
            signingConfig signingConfigs.debug
        }
    }
}

flutter {
    source '../..'
}

dependencies {
    implementation "org.jetbrains.kotlin:kotlin-stdlib-jdk7:$kotlin_version"
    implementation 'com.google.android.gms:play-services-location:21.0.1'
}

buildscript {
    ext.kotlin_version = '1.7.10'
    ext {
        compileSdkVersion = 33
        targetSdkVersion = 32
        minSdkVersion = 25
        appCompatVersion = "1.4.2"
        playServicesLocationVersion = "21.0.1"
    }
    repositories {
        google()
        mavenCentral()
    }
```

```
    dependencies {
        classpath 'com.android.tools.build:gradle:7.3.0'
        classpath "org.jetbrains.kotlin:kotlin-gradle-plugin:$kotlin_version"
        classpath 'com.android.support:multidex:1.0.3'
    }
}

allprojects {
    repositories {
        google()
        mavenCentral()
        maven { url
"${project(':flutter_background_geolocation').projectDir}/libs" }
        maven { url 'https://developer.huawei.com/repo/' }
        maven { url "${project(':background_fetch').projectDir}/libs" }
        configurations.all {
            resolutionStrategy {
                force "com.google.android.gms:play-services-location:21.0.1"
            }
        }
    }
}

rootProject.buildDir = '../build'
subprojects {
    project.buildDir = "${rootProject.buildDir}/${project.name}"
}
subprojects {
    project.evaluationDependsOn(':app')
}

tasks.register("clean", Delete) {
    delete rootProject.buildDir
}
```

ДОДАТОК Б Презентація

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ І ГОРЯ СІКОРСЬКОГО»
Кафедра Інженерії програмного забезпечення в енергетиці.

Програмний застосунок класифікації поведінки водія на дорозі

Виконала: студентка магістерського рівня 2 -го року навчання, групи ТВ -з21 мп
Ліла Владислава Олександрівна
Керівник: проф. Недашківський Олексій Леонідович

АКТУАЛЬНІСТЬ ТЕМИ ДОСЛІДЖЕННЯ

На даний час в Україні спостерігаються високі показники кількості випадків дорожньо-транспортних пригод. Загибель і травмування людей на дорогах створює напруження у сфері медичного обслуговування, а також завдає значної економічної шкоди для України. Як бачимо з вищенаведених тверджень, виникає гостра необхідність покращення ситуації із ДТП, яка склалась в даний період часу.

Наразі є різні додатки, що виконують автомобільну діагностику, спрощують керування автомобілем, аналізують ситуацію на дорогах. Однак багато з них доступні лише за кордоном, а частина додатків містить платний функціонал. В Україні поки що немає аналогів мобільних застосунків, які займаються саме класифікацією поведінки водія на дорозі, тому розробка такого проекту є актуальною зараз і буде сприяти покращенню ситуацій на дорогах.

ПОСТАНОВКА ЗАДАЧІ

- **Мета роботи:** зменшення кількості та наслідків дорожньо-транспортних пригод за допомогою розробки програмного продукту класифікації поведінки водія на дорозі за допомогою використання сенсорів мобільного телефону.
- **Об'єкт дослідження:** реєстрація, розпізнавання та класифікація поведінки водія на дорозі.
- **Предмет дослідження:** програмний застосунок класифікації поведінки водія на дорозі.

ЗАВДАННЯ ДОСЛІДЖЕННЯ

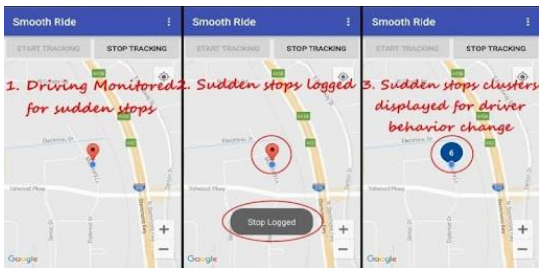
Розробка системи класифікації поведінки водія за допомогою сенсорів мобільного телефону.

Часткові наукові завдання:

1. Виконати аналіз даних і сенсорів, необхідних для розпізнавання поведінки водія.
2. Виконати аналіз алгоритмів та методів для класифікації поведінки водія.
3. Спроектувати архітектуру системи класифікації поведінки водія для мобільного додатку.
4. Визначити базу даних для розроблюваної системи та сторонніх є програмне забезпечення, що необхідно інтегрувати в систему.
5. Розробити систему класифікації поведінки водія на дорозі.

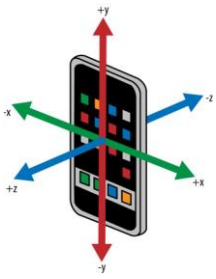
АНАЛОГІЧНІ ДОДАТКИ

Додаток Waze



Додаток Smooth Driver Monitoring

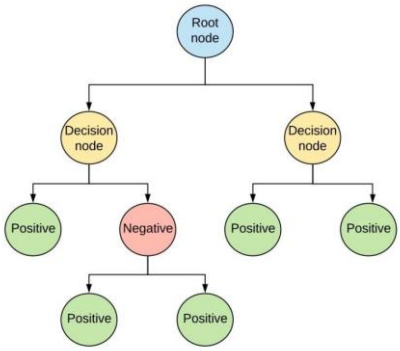
СЕНСОРИ, ВИКОРИСТАНІ В СИСТЕМІ



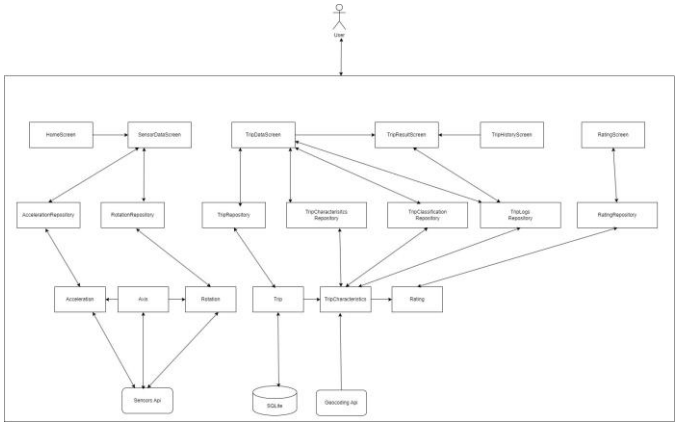
Accelerometer, Gyroscope Sensors

GPS Sensor

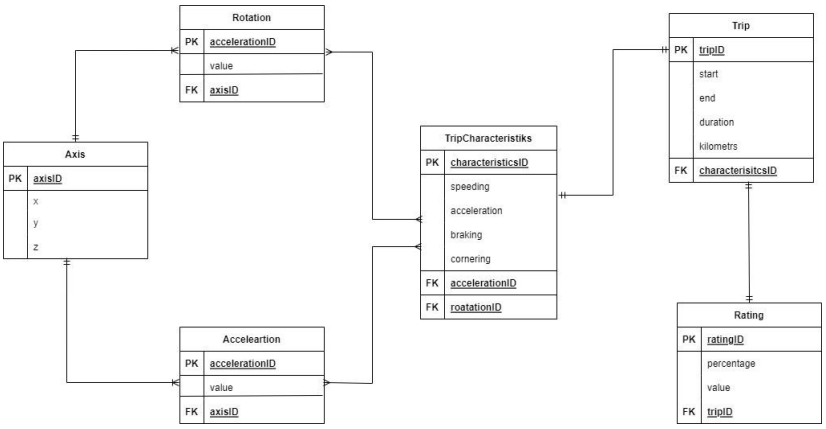
АЛГОРИТМ КЛАСИФІКАЦІЇ DECISION TREE



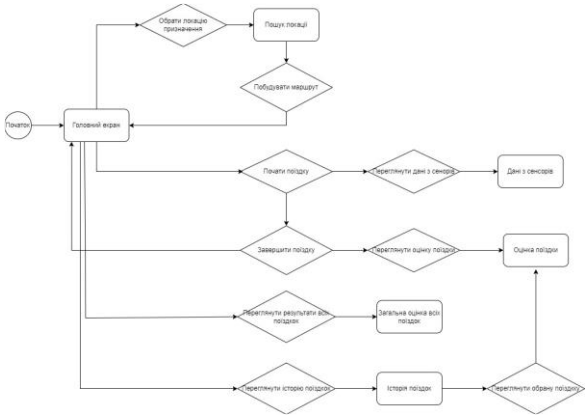
Огляд загальної схеми проекту «Класифікація поведінки водія на дорозі»



СТРУКТУРА БАЗИ ДАНИХ



ДІАГРАМА ЕКРАНІВ USER FLOW



ЗАСОБИ РОЗРОБКИ

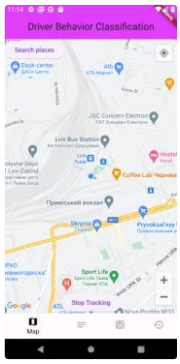
Dart — мова програмування для розробки веб- та мобільних додатків від компанії Google

Flutter – безкоштовний open-source мобільний фреймворк від компанії Google

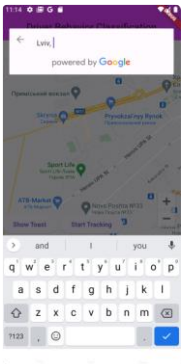
SQLite — легка швидка вбудована однофайлова СУБД

Android Studio — офіційне середовище розробки для платформи Android

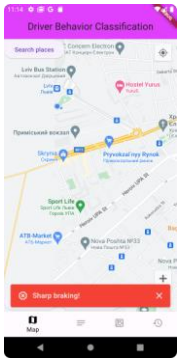
ГОЛОВНИЙ ЕКРАН ДОДАТКУ



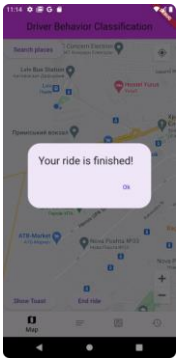
Мапа з маршрутом і рухом по ньому



Пошук локації при значення



Відображення попереджень і зауважень

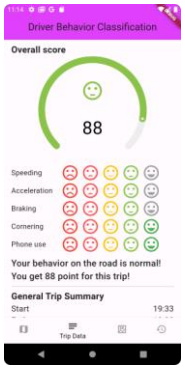


Завершення поїдки

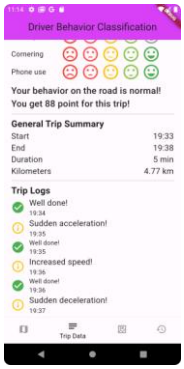
ЕКРАН З ДАНИМИ ІЗ СЕНСОРІВ

Відображення даних із сенсорів під час поїздки

ЕКРАН З ПІДСУМКАМИ ПОЇЗДКИ

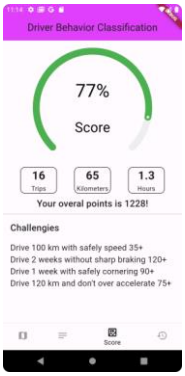


Оцінка поїздки, визначення поведінки водія і отриманих балів

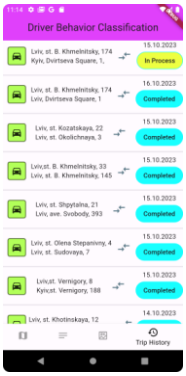


Загальна інформація і логи про поїздку

ІНТЕРФЕЙС В ЗАГАЛЬНИМ ФУНКЦІОНАЛОМ



Сумарні бали за всі поїздки і запропоновані виклики



Історія поїздок

ВИСНОВКИ

У ході виконання даної роботи було розроблено мобільний застосунок для класифікації поведінки водія на дорозі з можливістю відображення зауважень та попереджень у разі відхилення показників із сенсорів від норми та здійснення оцінки поїздки.

В результаті розроблене програмне забезпечення вирішує наступні проблеми:

- визначення місцезнаходження користувача за системою GPS;
- побудова маршрутів для навігації водія;
- збір даних із сенсорів під час руху транспортного засобу;
- відображення зауважень та попереджень під час поїздки;
- вивід даних про поведінку водія на дорозі;
- вивід оцінки поїздки відносно отриманих даних про поведінку водія;
- вивід загальної оцінки всіх поїздок протягом користування додатком.

Національний технічний університет України
«Київський політехнічний інститутімені Ігоря Сікорського»

National Technical University of Ukraine
"Igor Sikorsky Kyiv Polytechnic Institute"